BSD MAGAZINE

FOR NOVICE AND ADVANCED USERS

Kernel and syscalls
- Introduction

How to Upgrade OpenBSD
and Build a Kernel

Model View Whatever
Dolphin Smalltalk MVP

An Outsider Perspective
on Unikernels

Interview with Giuseppe Canale

VOL. 10 NO. 03
ISSUE 03/2016 (79)
1898-9144

SOMETHING XL
IS COMING IN APRIL

ENTERPRISE-CLASS HARDWARE, RUNNING
THE WORLD'S MOST POPULAR OPEN SOURCE
STORAGE OPERATING SYSTEM.

For more information on the FreeNAS Mini,
visit **iXsystems.com/mini** today.

Then, check back in April for something XL...

EDITORS' WORD

Dear Readers,

We hope you have all been doing well. This issue is focused on kernels. We will start with "An Outsider Perspective on Unikernels" by Hamza Sheikh. Russell Pavlicek will than explain you, how to "Understand Unikerlnes". What do you think about unikernels yourself?

Next, David Carlier will introduce you to "Kernel and Syscalls". We would also like to share with you the first module of David's online course "Developing FreeBSD Kernel Modules". We took down this course from the web page because materials were not completed, but the ones we have are very good and we would like to share them with you. If you are interested in learning "How to Upgrade OpenBSD and Build a Kernel", you have to take a look at Bob Cromwell's article. You will love it!

iXsystems shared an article about "Maxing Out Storage Performance with ZFS Caching" with us this month. Mark VonFange always has something ready for you ;)

The fifth installment of "Model View Whatever" series will be focused on "Dolphin Smalltalk MVP". Damian Czernous will evaluate "More vs less important view logic".

In this issue you will find one interview. We spoke with Giuseppe Canale about cyber security, Windows 10, being an expert and about open source, of course.

In the end, as always, is Rob's column and his point of view on the Panama Papers leak. What do you think about it? Has your country been mentioned in the many articles that have come out?

We would like to highly recommend you take a look at our online courses, especially if you liked the first module of "Developing FreeBSD Kernel Modules". We have three more online courses that you can still join and one that is slowly being prepared ;)

Also, we would like to thank all of you who contacted us after Marta's message to you. We have gotten so many responses! Next BSD Mag issue belongs to you!

Have a nice April!

Marta & BSD Team

MAGAZINE BSD

Editor in Chief:

Marta Ziemianowicz

marta.ziemianowicz@software.com.pl

Contributing:

David Carlier, Damian Czernous, Hamza Sheikh, Bob Cromwell, Russell Pavlicek Mark Von Fagne, Giuseppe Canale and Rob Somerville.

Top Betatesters & Proofreaders:

Annie Zhang, Denise Ebery, Eric Geissinger, Luca Ferrari, Imad Soltani, Olaoluwa Omokanwaye, Radjis Mahangoe, Mani Kanth, Ben Milman, Mark VonFange and David Carlier

Special Thanks:

Annie Zhang

Denise Ebery

DTP:

Marta Ziemianowicz

Senior Consultant/Publisher:

Paweł Marciniak

pawel@software.com.pl

CEO:

Joanna Kretowicz

joanna.kretowicz@software.com.pl

Publisher:

Hakin9 Media SK 02-676 Warsaw, Poland Postepu 17D Poland worldwide publishing editors@bsdmag.org www.bsdmag.org

Hakin9 Media SK is looking for partners from all over the world. If you are interested in cooperation with us, please contact us via e-mail: editors@bsdmag.org.

CONTENTS

CONTENTS

BSD Certification

The BSD Certification Group Inc. (BSDCG) is a non-profit organization committed to creating and maintaining a global certification standard for system administration on BSD based operating systems.

WHAT CERTIFICATIONS ARE AVAILABLE?

BSDA: Entry-level certification suited for candidates with a general Unix background and at least six months of experience with BSD systems.

BSDP: Advanced certification for senior system administrators with at least three years of experience on BSD systems. Successful BSDP candidates are able to demonstrate strong to expert skills in BSD Unix system administration.

WHERE CAN I GET CERTIFIED?

We're pleased to announce that after 7 months of negotiations and the work required to make the exam available in a computer based format, that the BSDA exam is now available at several hundred testing centers around the world. Paper based BSDA exams cost $75 USD. Computer based BSDA exams cost $150 USD. The price of the BSDP exams are yet to be determined.

Payments are made through our registration website:
https://register.bsdcertification.org//register/payment

WHERE CAN I GET MORE INFORMATION?

More information and links to our mailing lists, LinkedIn groups, and Facebook group are available at our website:
http://www.bsdcertification.org

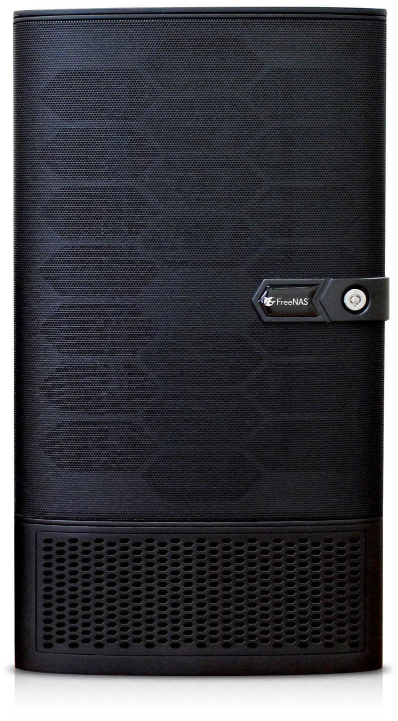
Registration for upcoming exam events is available at our registration website:
https://register.bsdcertification.org//register/get-a-bsdcg-id

New FreeBSD Foundation Logo

The FreeBSD Foundation is pleased to announce the debut of our new logo and website, signaling the ongoing evolution of the Foundation identity and ability to better serve the FreeBSD Project. Our new logo was designed to not only reflect the established and professional nature of our organization, but also to represent the link between the Project and the Foundation, and our commitment to community, collaboration, and the advancement of FreeBSD.

https://www.freebsdnews.com/2016/03/25/new-freebsd-foundation

FreeNAS Mini XL Now Available



New and Improved Device Offers Greater Storage Capacity, Performance Than Its Predecessor

SAN JOSE, CA--(Marketwired - April 04, 2016) - iXsystems, the server, storage, and open-source technology company that develops FreeNAS, the world's most popular storage OS, today introduced the FreeNAS Mini XL Storage Device. The FreeNAS Mini XL is based on the same proven hardware used by the smaller FreeNAS Mini, but offers greater storage capacity and performance. Available today, the FreeNAS Mini XL can be purchased at iXsystems.com or from Amazon.

Storage solutions like the FreeNAS Mini XL have become increasingly important to users as technologies that allow for content capture and creation become mainstream and intensive. More than ever, customers value solutions that allow for large amounts of storage that are fast and secure.

The FreeNAS Mini XL will be offered in multiple configurations ranging from Diskless to 48TB. Users configuring a Mini XL build through iXsystems.com have a variety of customizable options, including: read and write cache drives, different ECC DRAM capacities, and of course, storage configurations up to 48TB.

"The FreeNAS Mini XL satisfies a long-standing customer need for twice the storage in a footprint not much larger than our highly popular FreeNAS Mini product," said Jordan Hubbard, CTO, iXsystems. "Our customers are capturing 4K video, curating massive media libraries, and want a storage option that is fast and reliable. The Mini XL based on FreeNAS gives them that solution."

FreeNAS Mini XL Key Features:

- Power efficient Eight-Core 2.4GHz Intel CPU, drawing only 17 Watts
- Hardware-accelerated ZFS disk encryption
- 32GB of ECC memory, upgradeable to 64GB
- Hot swap functionality with 8 tool-less, front loading drive bays
- Capable of saturating dual Intel Gigabit LAN Ports, but can be upgraded to 10GbE if needed
- Dedicated Remote Hardware Management Interface (IPMI) or option to share through on-board LAN

- Latest FreeNAS version installed and configured on dedicated, internal flash device
- 3 x USB 2.0 Ports (2 front, 1 rear), 1 x Serial Port (DB9), 1 x VGA
- Physical security: locking front bezel, latchable drive trays, and chassis lock eyelet (for cable or padlock)

http://www.ixsystems.com/

“Mr. Linux” Of Oracle Joins Microsoft To Lead Open Source And Linux Efforts

Microsoft’s HR department has poached the man who turned Oracle into a Linux-dominated company. Wim Coekaerts, commonly known as Oracle’s “Mr. Linux”, has joined Redmond folks to head up its Linux and open source efforts.

This is an established fact that Microsoft loves Linux and open source technologies. The recent developments at Build Developer Conference 2016 will soon allow developers to use Ubuntu on Windows 10.

In order to hasten the progress of Linux and open source projects at Redmond, the company’s HR department has stepped up its game. The software giant has reportedly poached Wim Coekaerts, commonly known as Oracle’s “Mr. Linux”.

In the tech world, Coekaerts is well-known for his contribution to Linux projects and bringing the open source technology to Oracle. He also turned Oracle into a Linux distro-maker with the launch of Oracle Linux for the enterprise.

Talking to ZDNet, Mike Neil, Microsoft’s Corporate VP of the Enterprise Cloud has confirmed this news.

“Wim Coekaerts has joined Microsoft as Corp VP of Open Source in our Enterprise Cloud Group. As we continue to deepen our commitment to open source, Wim will focus on deepening our engagement, contributions and innovation to the open-source community,” he said.

Microsoft became involved with Linux since Satya Nadella became CEO, going as far as creating a custom Linux-based software for networking and Azure.

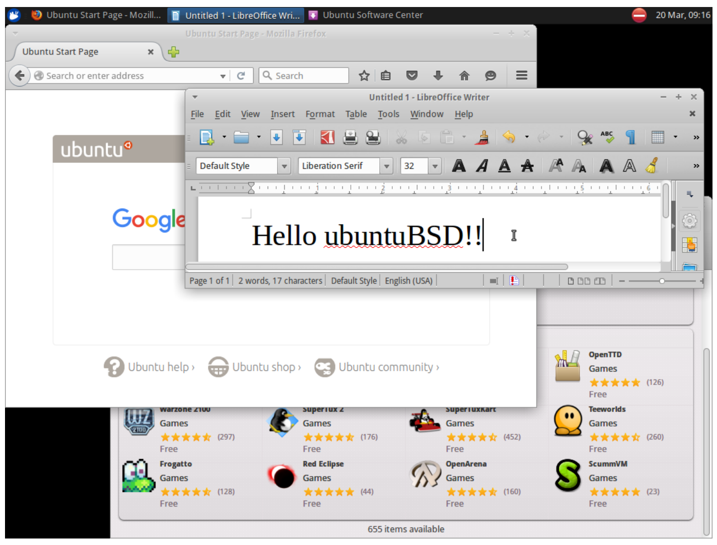
The company hasn’t said what parts of its open source efforts will be taken care of by Coekaerts, but it’s important to note that he is the person who turned Oracle into a company dominated by Linux-based technologies.

http://fossbytes.com/mr-linux-wim-coekaerts-oracle-joins-microsoft-open-source/

UbuntuBSD now available

UbuntuBSD has been made available, and is currently in its BETA stage. This project aims to provide the Ubuntu user experience on top of the FreeBSD kernel, with the goal of moving away from systemd.

The ease and familiarity of Ubuntu with the rock-solid stability and performance of the FreeBSD kernel.



This project is in BETA stage! It is production-ready in most cases but you could easily find some bugs. You have been warned.

Features

- Versatile text based installer
- Xfce desktop included
- Ubuntu based
- Debian based
- FreeBSD based
- Suitable for desktops and servers alike
- ZFS support completely integrated

https://www.freebsdnews.com/2016/03/25/ubuntubsd/

https://sourceforge.net/projects/ubuntubsd/

Save it, devs. Red Hat doesn't want your $99 for RHEL

Free pilot's license for immutable infrastructure nuts

Red Hat has cut the $99 price of its Linux developer subscription to zero, for penguins building cloud microservices using containers.

The company today is expected to start giving away its Red Hat Enterprise Linux (RHEL) subscription for free as part of the existing Red Hat Developer Program.

The free license runs in tandem with the existing $99 developer license for those already paying, with Red Hat assessing whether it should continue charging.

It follows the introduction of a free developer license for the JBoss application server, FUSE, Drools and BPM suite 18 months ago.

Like these, the RHEL license applies only to development, not production environments. Unlike these, it's RHEL that's turned Red Hat into a profitable Linux vendor and the first past $1bn and then, last year, $2bn in revenue.

While this targets devs, however, Red Hat's introduced the free license to ensure its Linux becomes a player in continuous lifecycle management of cloud services.

Red Hat wants to ensure RHEL gets used from dev and test through to production rather than, as can happen now, have devs start out using free CentOS or Fedora and then – potentially – switch to RHEL for production or – more likely – stay on the free stuff.

It's that need to switch that introduces the risk of coding breaks, and that introduces migration hassles and is a conclusive factor in preventing lifecycle management.

That's a no-no for today's meme du-jour of immutable infrastructure – servers build using code that you don't alter, but are built for purpose and get dumped if, or when, they're done with or get broken. Immutable infrastructure cannot work if there exist minute variations in different servers or different versions of the same server.

One operating system all the way down the line is one possible way to avoid this.

Announcing the free license, Red Hat also pointed out it's targeting those coding for Linux from Windows machines and Mac.

Red Hat buddied up with Microsoft in November last year on cloud. RHEL, JBOSS and middleware would run natively on Microsoft's Azure cloud with plans for "full access to .NET across Red Hat technologies," Mower said.

RHEL became the "primary development and reference operating system for .NET Core on Linux." ®

http://www.theregister.co.uk/2016/03/31/red_hat_rhel_free_dev_license/

NEWS

Microsoft to make Xamarin tools and code free and open source

Now Included for FREE

Announcing
Xamarin for Visual Studio
Visual Studio Enterprise
Visual Studio Professional
Visual Studio Community Edition

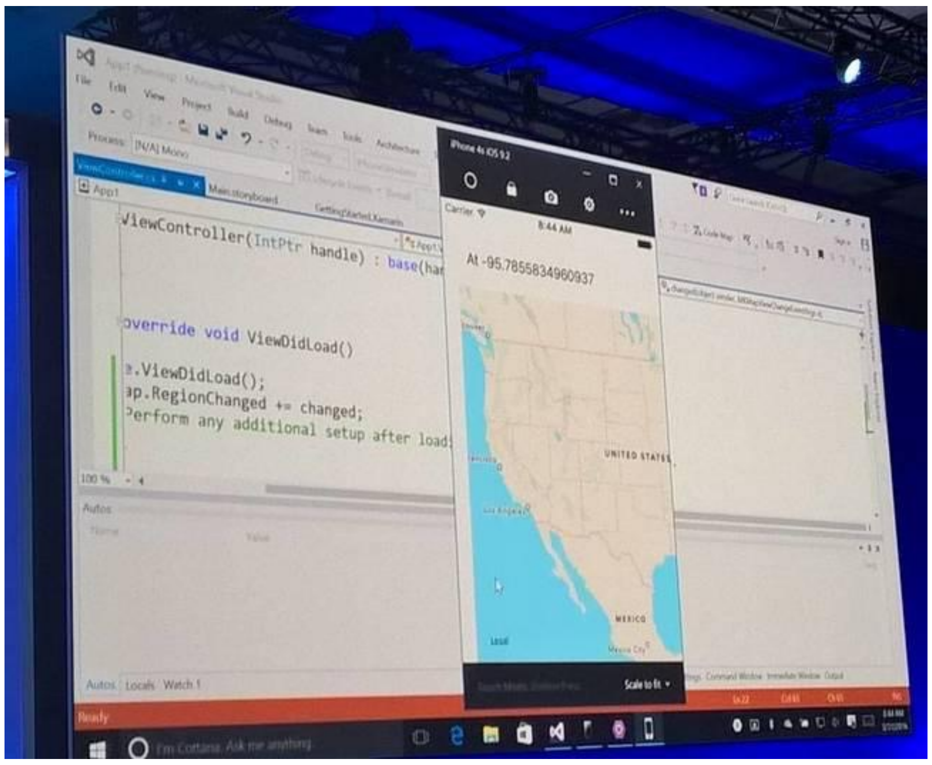
Xamarin casts off commercial roots

BUILD2016 Microsoft will make Xamarin tools and code, which enable compilation of Mac, Android and iOS applications using C#, free and open source, said corporate VP Scott Guthrie at the company's Build conference under way in San Francisco.

Xamarin's origins are with Mono, a project created by Miguel de Icaza as an open source and cross-platform implementation of Microsoft's C# language and .NET Framework. In 2011, Xamarin was cofounded by de Icaza and Nat Friedman, taking the Mono platform and making it a compiler for mobile apps targeting iOS and Android.

Xamarin platform will now be free for Visual Studio users, including the free Community edition as well as Professional and Enterprise versions – drawing a big cheer from Build attendees.

In addition, Guthrie said that Xamarin's code will be open source, under the management of the .NET Foundation, home of Microsoft's other open source developer projects including ASP.NET and .NET Core.



```
ViewController(IntPtr handle) : base(ha

override void ViewDidLoad()

e.ViewDidLoad();
ap.RegionChanged += changed;
Perform any additional setup after load
```

Xamarin's iOS emulator remoted to Windows for debugging

Miguel de Icaza also took the stage at Build, to show off a new feature: the ability to remote the iOS emulator to Windows in order to test and debug an iOS application in Visual Studio without leaving the Windows desktop. A Mac is still required, since part of the compiler runs on Apple's OS X.

Separately, Mono has been relicensed under the MIT license, which means it can now be used freely even in scenarios which would previously have required a commercial license. Mono is also being contributed to the .NET Foundation.

Microsoft's Xamarin strategy will boost usage of the tools and will further the company's ambition, announced yesterday at Build, to make Windows an all-purpose development platform.

That said, the preferred operating system of many mobile developers is the Mac, and while Microsoft's announcements will do much to keep its own developer community using Windows, winning back Mac-using developers will be more challenging.

Here's the thing though: it was open source guy de Icaza who took Xamarin closed source, and now it is Microsoft returning it to the community. Surprising times. ®

http://www.theregister.co.uk/2016/03/31/xamarin_tools_code_free_and_open_source/

The OpenBSD Foundation 2016 Fundraising Campaign

The OpenBSD Foundation needs your help to achieve our fundraising goal of $250,000 for 2016.

Reaching this goal will ensure the continued health of the projects we support, will enable us to help them do more, and will avoid the distraction of financial emergencies that could spell the end of the projects.

2015 was a good year for the foundation financially, with funding coming almost equally from corporate and community donations. While the total was down significantly after 2014's blockbuster year, we again exceeded our goal.

As an individual or corporation, the best kind of donation we can receive is a recurring donation. This allows longer term planning on our part, instead of hoping for one time cash infusions. The easiest way for an individual to support us in this way is a recurring Paypal donation, which is our preference.

Donations to the foundation can be made on our Donations Page. We can be contacted regarding corporate sponsorship at fundraising@openbsdfoundation.org

Hadoop rebels unleash spec to battle the Cloudera/MapR empire

https://bsdmag.org/openbsd_found/

ODPi publishes runtime spec and test suite

ODPi, the group formerly known as the Open Data Platform initiative and set up last year as an attempt to standardise Hadoop applications, has published its first runtime specification.

Backed by Hortonworks but kicked into the corner by heavyweights MapR and Cloudera, ODPi was set up last year to try and make sure applications would work

Instead of collapsing, however, the group quietly worked on a runtime specification accompanied by a test suite.

The ODPi technical working group says its objectives are:

- For consumers: ability to run any "ODPi-compatible" software on any "ODPi-compliant" platform and have it work.
- For ISVs: compatibility guidelines that allow them to "test once, run everywhere."
- For Hadoop platform providers: compliance guidelines that enable ODPi-compatible software to run successfully on their solutions. But the guidelines must allow providers to patch their customers in an expeditious manner, to deal with emergencies.

The runtime spec, at Github, doesn't preclude applications from calling private interfaces, but the authors note that customisation breaks the "test once, run everywhere" model.

The current spec requires the Apache Hadoop 2.7 branch as a base spec. The "minimum native build" specifications set out requirements for Kerberos, Java and OS requirements, and the GZip and Snappy codec compression libraries.

Apache Big Top is used for packaging, testing and configuration, and there are "guidelines on how to incorporate additional, non-breaking features, which are allowed provided source code is made available through relevant Apache community processes."

Sandbox images for testing are also at Github. ®

http://www.theregister.co.uk/2016/03/29/hadoop_rebels_unleash_spec_to_battle_the_clouderamapr_empire/

Open Source Hybrid Cloud Sends Red Hat Past $2B Revenue Mark for 2016

If there was any question about just how profitable an open source business can be, Red Hat (RHT) answered it this week by releasing financial reports that show it has become the first open source company to cross the $2B revenue threshold.

The company's latest financial report, released March 22, shows revenue of $544M for the last quarter and $2.05B for fiscal year 2016 overall. That's up from $1.79B in annual revenue for FY 2015.

The increase is not at all a bad showing for a period during which the global economy remained sluggish and American companies like Red Hat faced a tough international market due to the strong dollar.

Red Hat said the growth reflected the success of its open source hybrid cloud products in particular. "Enterprises increasingly adopting hybrid cloud infrastructures and open source technologies drove our strong results," said Jim Whitehurst, the company's CEO. "Customers are demanding technologies that modernize the development, deployment and life-cycle management of applications across hybrid cloud environments. Many are relying on Red Hat to provide both the infrastructure and the application development platforms to run their enterprise applications consistently and reliably across physical, virtual, private cloud and public cloud environments."

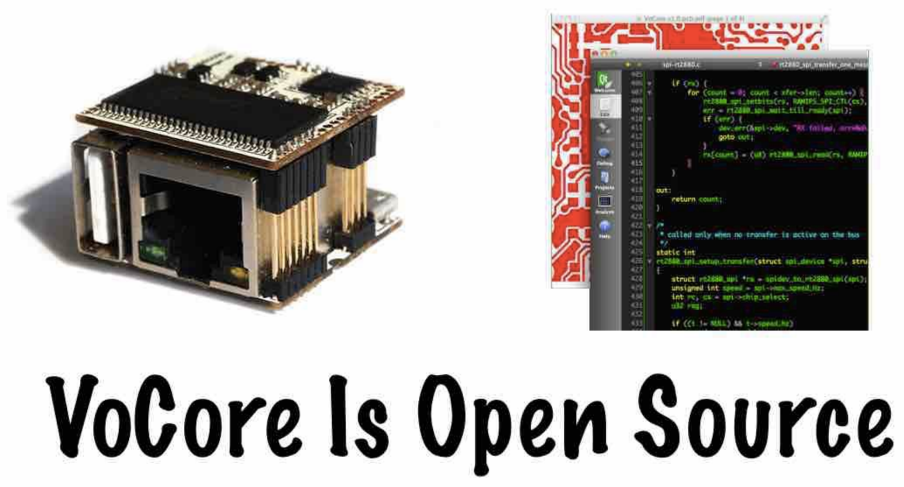
Going forward, Red Hat projects revenue of between $2.380B and $2.420B for the coming fiscal year.

It's no secret that businesses can succeed by selling software whose source code is given away for free, or services related to it. Companies like Cygnus Solutions (which later merged with Red Hat) were profiting from free software as early as the 1980s. But Red Hat's crossing of the $2B threshold is a new milestone for the open source ecosystem.

The figure might not be significant for larger vendors that don't focus exclusively on open source, but for a company that has invested everything in open source, it's a big deal. It's also good news for the open source sector of the channel as a whole, which can continue to count on strong performance at companies like Red Hat, which funds a great deal of open source development, to keep open source projects well-heeled.

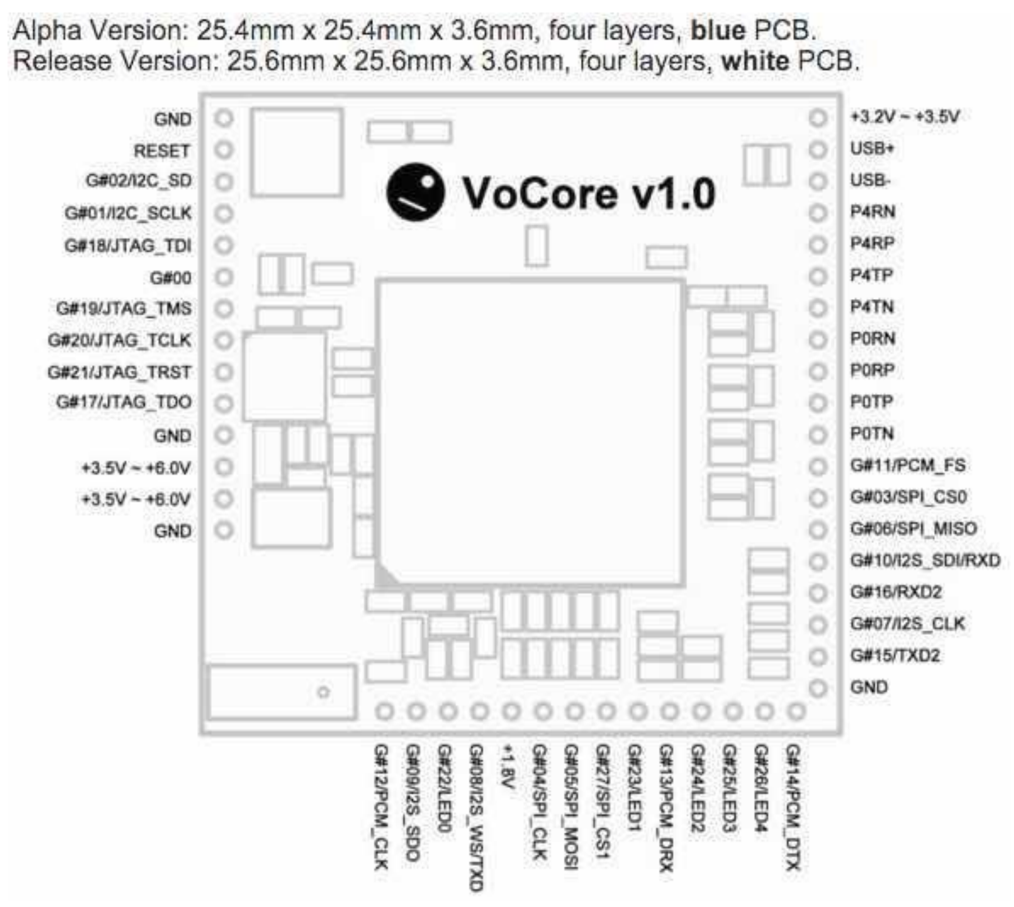
http://thevarguy.com/open-source-application-software-companies/open-source-hybrid-cloud-sends-red-hat-past-2b-revenue-ma

VoCore: A Cheap And Coin-sized Linux Computer With Wi-Fi



VoCore is an open hardware that runs OpenWRT Linux. This tiny computer comes with Wi-Fi, USB, 20+ GPIOs that will help you to make a smart home automation system or use it in other embedded projects. Read more to know about this device and grab one for yourself.

With each passing day, mini computer boards are getting more and more popular. Single board computers like Raspberry Pi, CHIP, OrangePi, etc., are being endorsed by makers and DIY enthusiasts to create new innovations. However, if you are looking for an even smaller Linux computer, VoCore is the perfect device for you.



At fossBytes, we love open source technologies and so do the makers of VoCore. It's an open source hardware that runs OpenWrt on top of Linux. With this mini Linux machine, you can fork your own Wi-Fi router, make smart home automation systems, invent a new device, build a motherboard and whatnot.

In simpler words, you can use it as a standalone device or use it as an embedded component of a larger system. It comes with included Dock that extends the USB and Ethernet ports to enhance its functionality.

Specifications of VoCore Linux computer:

This coin-sized computer is powered by 32MB SDRAM, and 8MB SPI Flash. It provides many interfaces like 10/100M Ethernet, USB, UART, I2C, I2S, PCM, JTAG and over 20 GPIOs.

- OpenWRT Linux
- Onboard Wi-Fi adapter
- Integrates an 802.11n MAC, baseband, radio, FEM & 5-port 10/100Mbps Ethernet switch
- Processor: Ralink/Mediatek 360 MHz RT5350 MIPS

Here's the pin map of VoCore:

An open source software and hardware of VoCore means that you will also get its full hardware design and source code. This will allow you to control every part of this tiny Linux computer.

http://fossbytes.com/vocore-cheap-linux-computer-wi-fi-openwrt/

Great Specials

On FreeBSD® & PC-BSD® Merchandise

Give us a call & ask about our
SOFTWARE BUNDLES
1.925.240.6652

$39.95
FreeBSD 9.1 Jewel Case CD Set
or FreeBSD 9.1 DVD

$29.95
PC-BSD 9.1 DVD

$49.95 Save a BUNDLE!
The PC-BSD 9.0 Users Handbook
PC-BSD 9.1 DVD

$99.95
The FreeBSD CD **or** DVD Bundle

Inside each CD/DVD Bundle, you'll find:
FreeBSD Handbook, 3rd Edition
Users Guide FreeBSD Handbook, 3rd Edition, Admin Guide
FreeBSD 9.1 CD or DVD set
FreeBSD Toolkit DVD

Stylish Dress Attire
Look Your Professional Best

Comfy Apparel
Stay Warm in Zip Ups & Pullovers

T-Shirts
Lots of Styles to Choose From

FreeBSD 9.1 Jewel Case CD/DVD	$39.95

CD Set Contains:

- **Disc 1** Installation Boot LiveCD (i386)
- **Disc 2** Essential Packages Xorg (i386)
- **Disc 3** Essential Packages, GNOME2 (i386)
- **Disc 4** Essential Packages (i386)

FreeBSD 9.0 CD	$39.95
FreeBSD 9.0 DVD	$39.95

FreeBSD Subscriptions

Save time and $$$ by subscribing to regular updates of FreeBSD

FreeBSD Subscription, start with CD 9.1	$29.95
FreeBSD Subscription, start with DVD 9.1	$29.95
FreeBSD Subscription, start with CD 9.0	$29.95
FreeBSD Subscription, start with DVD 9.0	$29.95

PC-BSD 9.1 DVD (Isotope Edition)

PC-BSD 9.1 DVD	$29.95
PC-BSD Subscription	$19.95

The FreeBSD Handbook

The FreeBSD Handbook, Volume 1 (User Guide)	$39.95
The FreeBSD Handbook, Volume 2 (Admin Guide)	$39.95

The FreeBSD Handbook Specials

The FreeBSD Handbook, Volume 2 (Both Volumes)	$59.95
The FreeBSD Handbook, Both Volumes & FreeBSD 9.1	$79.95

PC-BSD 9.0 Users Handbook	$24.95
BSD Magazine	$11.99
The FreeBSD Toolkit DVD	$39.95
FreeBSD Mousepad	$10.00
FreeBSD & PCBSD Caps	$20.00
BSD Daemon Horns	$2.00

Bundle Specials!
Save $$$

Just Plain Fun
Mousepads & Novelty Horns

BSD Magazine
Available Monthly

FreeBSD Mall

For even **MORE** items
visit our website today!
www.FreeBSDMall.com

An Outsider Perspective on Unikernels

by Hamza Sheikh

I have read *The Rise and Fall of the Operating System* by Dr. Antti Kantee as well as Unikernels are Unfit for Production by Bryan Cantrill. As an outsider, I have much to learn about Unikernels. I also don't have a horse in the race yet. For me, this is a very academic debate on the pros and cons, today and in the next decade, of Unikernels.

Kantee writes:

"Technology should encapsulate complexity and be optimized for the common case, not for the worst case, even if it means we, the software folk, give up the illusion of being in control of hardware."

This is a hard idea to digest for me. In my Quality Assurance roles, I feel a deep desire to explore the worst case scenarios to get an understanding of when and how systems fail and their impact when they fail. For me, the happy path, the "common case" as Kantee writes, is to make sure systems don't regress during active development. The common case deserving more optimization goes against the grain of my approach.

Systems don't exist in isolation anymore. In a networked world, there are many more chances an individual system will fail because it interacts with the outside world. I'm inclined to optimize for recovering gracefully and also providing enough information to figure out the why and how of the failure.

Kantee continues:

"In other words, since the operating system does not protect the user from evil or provide powerful abstractions, it fails its mission in the modern world. Why do we keep on using such systems?"

This is a very interesting observation. Cantrill makes a similar point about Unikernels,

"The organs that provide this kind of functionality have been deliberately removed from unikernels in the name of weight loss; any unikernel that provides sufficiently sophisticated debugging tooling to be used in production would be violating its own dogma."

They're both making the point that since a particular system does not adhere to its own purpose or design it crosses the boundary into hypocrisy. Are we to throw out entire bodies of work because they are not pure enough?

Kantee concludes:

"Minimally implemented application support is a few thousand lines of code plus the drivers, as we demonstrated with the Rumprun unikernel. Therefore, there is no reason to port and cram an operating system into every problem space. Instead, we can split the operating system into the "orchestrating system" (which also has the catchy OS acronym going for it) and the drivers. Both have separate roles. The drivers define what is possible. The orchestrating system defines how the drivers should work and, especially, how they are not allowed to work. The two paths should be investigated relatively independently as opposed to classic systems development where they are deeply intertwined."

This conclusion is important to keep in mind when reading Cantrill's post or thinking about Unikernels in general.

Unikernels are a think-outside-the-box approach to solving the problem of adapting to the various use cases (bare metal, cloud, mobile, desktop, etc.) by doing the minimum work necessary to get things running. As Cantrill points out, this sacrifices a lot of the tooling and experience gained in operations used to solve the same problems today.

As evidenced by Rumprun, running existing applications unmodified requires sacrifices from the Unikernel. For example, POSIX stuff is still required for many applications. In addition, in a perfect Unikernel-only world, a vast majority of applications will need to be rewritten. As Cantrill points out, "There are no processes" and "Good news: apps kinda work! Bad news: did we mention that they need to be ported?"

Who out there is willing to learn a new paradigm of deploying applications that also requires them to rewrite them? Windows XP is still in use throughout the world even though it has been End of Life for a while now. Change is not easy, especially when there are huge monetary investments to be protected.

This leads me to believe that Unikernels will be successful in niches. The size of these niches may be significant and that may attest to their success with time. Nevertheless, I don't see them taking over the world in the next decade.

Unikernels also throw out the baby with the bath water. Instead of building on the advances made in Operating Systems (OS) and tools, they rely on every application deciding how much of the OS work it is willing to reimplement.

For example, debugging, monitoring, and other operational considerations are now the responsibility of each application, instead of the same applications using specialized work in the OS ecosystem in those areas.

When I look at the maker and Internet of Things world, the world of Raspberry Pis and Arduinos, I like the idea of Unikernels a lot. They potentially provide only the things I need to get my single application running. This is why I think Unikernels are bound to shine in highly specialized niches. They are not going to replace my OS X anytime soon.

Many of us have to run multiple applications simultaneously, many of which share libraries. These libraries are abstractions built upon other abstractions. Unikernels aim to get rid of these layers of abstractions and interfaces and provide a single-layered surface where, as I understand it, it's a giant ball of code compiled together. The application is the OS and the OS is the application. I still can't wrap my head around the changes required in software development practices with this sea change.

Unikernels have an uphill battle in convincing software practitioners that this radical idea will provide handsome dividends at some point. They are at an unfortunate disadvantage because they either have to win at everything or they will be declared a failure. They are also burdened with casting off any crutches to make current applications run as soon as they can or else the purity of the entire idea will be questioned. It will be said, "If they need this idea and that thing and that other stuff from traditional OSes what's the big idea here anyways?"

I'm taking a wait and watch approach. I would love to be proven wrong in my assumptions and predictions. What I do know for sure is that as Unikernels mature and gain wider adoption, the debate will not only intensify but we will also witness their evolution. We will get a better idea of how some theoretical things are executed in practice and how well they are received. I look forward to following this technology.

About the Author:

A Linux escapee and FreeBSD user of less than a year. Started using BSDs with pfSense in 2007 and loved it. Day job involves a lot of Python and test automation. Evenings and weekends are devoted to family, FreeBSD, and Erlang.

@aikchar

Understanding Unikernels

by Russell Pavlicek

When we describe a typical operating system kernel on a typical machine (be it physical or virtual), we are normally talking about a distinct piece of software which runs in a separate processor mode (kernel mode) and address space from the rest of the software running on that machine. This operating system kernel generally provides critical low-level functions which are leveraged by the other software installed on the box. The kernel is generally a generic piece of code which is trivially tailored (if at all) to the application software stack it is supporting on the machine. This generic kernel normally provides a wide range of rich functions, many of which may be unneeded by the particular applications it is being asked to support.

In fact, if you look at the total software stack on most machines today, it is often difficult to figure out just what application will be run on that machine. You are likely to find a wide swath of hundreds, if not thousands, of low-level utilities, plus multiple databases, a web server or two, and a number of specialized application programs. The machine may actually be charged with running a single application, or it may be intended to run dozens simultaneously. Careful analysis of the startup scripts will yield hints as to the final solution set which will be run on the machine, but it is far from certain, as a suitably privileged user may elect to invoke any of a number of applications present on the box.

The Unikernel Difference

The footprint of a unikernel-based machine, however, is quite different. A unikernel's role in a machine (or virtual machine image) is similar to that of other kernels, but the implementation parameters are significantly different.

For example, an analysis of a unikernel-based machine's code does not suffer from the ambiguity of most other software stacks. When you look at a unikernel system, you will find one and only one application present. The multiple applications of a standard software stack are gone, as are the aforementioned plethora of generic utilities and support functions.

But the trimming of excess doesn't stop there. Not only is the application stack trimmed to the bone, but the operating system functions are likewise reduced. For example, multi-user support is gone. Multiple process support is gone. Advanced memory management is gone.

Think that's radical? Consider this: the whole notion of a separate operating system layer is gone as well! No longer is there a separate address space for the kernel and a separate address space for the application. Why? Because the kernel functions and the application are now part of the same program. In fact, the entire software stack is comprised of a single software program, providing all needed application code and operating system functions. And, if that weren't enough, the operating system functions contained in the unikernel provide only those functions needed to power the application in question – all other unneeded operating system functions have been removed entirely.

A Software Stack Reflecting The Realities of a New Century

This singularity of purpose is the radical rethink behind unikernels. For decades in this industry, we have worked with the concept that the best architecture for any machine is to start with a generic multiuser operating system base, load on a wide array of useful utilities, add every application we might possibly want to use, and then top it off with some type of packaging software to manage this entire mess.

Thirty-five years ago, this approach was entirely sensible. Hardware was expensive. Virtualization options were very limited or unavailable. Security was limited to making sure the guy seated next to you in the computing center wasn't watching you type your password. A single machine had to handle many users running many applications at the same time in order to be cost effective. When I was in college (a millennium or two ago), before the arrival of the personal computer, the college computer center had one insanely expensive machine (by today's standards) – a DEC PDP-11/34a, for the gray-headed who might care – equipped with 248 kilobytes (no, that's not a typo) of usable memory and 25 megabytes of disk space to support an entire campus of computer science, engineering, and mathematics students. That single machine had to service every function which a couple hundred students could think of in a given semester.

Compare that ancient dinosaur of computing history to a modern smart phone and you discover that the phone has multiple orders of magnitude more computing power than that machine had. So why are we still creating machine images using the same rules we used back in the computing stone age? Doesn't it make sense to rethink the software stack to match the new realities of computing?

In the modern world, hardware is cheap. Virtualization is omnipresent and efficient. And just about every computing device is connected to a vast worldwide network of potentially malicious hackers. Think about it: a DNS server really doesn't need gigabytes of bloat to do its job. An app server doesn't really need to have a thousand utilities available for the use of the desperado who just exploited a hole which gave him virtual command line access. And a webserver doesn't need to validate the command line logins of 500 different timesharing users. So why are we still using an archaic software stack concept which supports all these undesirable scenarios?

The Brave New World of the Unikernel

So what should a modern stack look like? How about this: a singular application image, virtualized, highly secure, ultra-lightweight, with incredibly fast startup times. That's something that a unikernel can deliver. Let's break it down:

A Singular Image: The hundreds of utilities and masses of applications layered on top of a generic kernel is replaced with a single executable containing all the needed application and operating system code in a single image. It contains nothing more than what is needed.

Virtualized: Just a few years ago, you'd be lucky to fit anything more than a handful of virtual machines on a single server. The limits in memory in hardware combined with the old-school, memory-hungry software stack simply didn't allow room for too many VMs to coexist on a single server at one time. With today's beefy servers equipped with many gigabytes of memory, we no longer need to settle for a handful of VMs per server. If each VM image were small enough, we could run hundreds – or even thousands – of VM appliances simultaneously on a single server.

Secure: In the age of the cloud, we find malicious hackers routinely penetrating servers everywhere, even the servers of the largest corporations and government agencies. Such violations frequently exploit flaws in a network-enabled service to break into the lower levels of the software stack. From there, the malefactor can use utilities and other programs present on the box to commit their nefarious acts. In a unikernel stack, there are no other pieces of software to assist the malevolent hacker. The hacker has to be bright enough to penetrate the application, but then be brighter still to do evil things without any resident tools to leverage. It does not make the software completely secure, but it does raise the security bar significantly – and that's an advancement which is long overdue in the cloud.

Ultra-lightweight: A normal VM could consume gigabytes of disk space and memory just to make a few services available to the network. With a unikernel, it's possible to gut those requirements. For example, MirageOS (a popular unikernel system) has a functional DNS server which uses 449 kilobytes of disk space – yes, less than half of a megabyte.

ClickOS, a network appliance unikernel system from NEC labs, has actual network devices that use just 6 megabytes of memory while successfully processing over 5 million packets per second. These are by no means atypical examples of unikernel-based devices. Given the diminutive footprint of unikernels, the notion of stacking hundreds or thousands of these tiny VMs on a single host server no longer seems far-fetched.

Fast startup: Normal VMs can take serious time to boot up. It can take a minute or more on modern hardware for a full operating system and application stack to come online. However, this is not the case for a unikernel-based VM. Most unikernel VMs boot in less than a tenth of a second. For example, ClickOS network VMs have been documented to boot up in under 30 milliseconds. That's quick enough to birth a responding VM as soon as the service request appears on the network (which is one of the things which the Jitsu project does – see http://unikernel.org/files/2015-nsdi-jitsu.pdf).

But Don't Containers Already Do This?

Containers go a long way toward creating lightweight, fast VMs. But containers still rely on a shared beefy operating system underneath the covers. That is still a lot to lock down from a security standpoint. And it is clear that we need to step up our security in the cloud, rather than pursue the same old security methods which are rapidly becoming problematic in the cloud. Beyond that, the final footprint of unikernels is still smaller than what containers can provide. So containers move in the right direction, but unikernels manage to go farther in the direction which we need in tomorrow's cloud.

So How Do Unikernels Work?

As mentioned earlier, a traditional machine is built from the bottom up: you select a general-purpose kernel, add a slew of utilities, and then add the applications. Unikernels are the exact opposite: they are built top-down. Focus on the application you need to run and then add just enough operating system functions to make it come to life.

Most unikernel systems rely on a compile-and-link system which takes the source code of the application and links in libraries which provide only those operating system functions that application requires, resulting in a single compiled image that can be run in a VM as-is without any additional software.

How Do You Debug the Result?

Since there is no operating system or utilities in the final product, most unikernel systems use a staged approach to development. Often, a compile while in the development phase will yield an executable suitable for testing on Linux or other Unix-like operating system. This executable can be run and debugged like any standard program. Once you are satisfied with the result, you recompile with the production switch turned on, creating the final image meant to be run standalone in a VM.

The lack of debugging tools on the production machine isn't as awkward as it may sound at first. The limitations on debugging a production image are mitigated by the fact that in most organizations, developers are not allowed to actually debug on a production machine anyway. Instead, they gather up logs and other information, recreate a failure on a development platform, make corrections, and redeploy the result. That exact sequence is readily available in the unikernel world. You just need to make sure that your production image will produce enough logging output to facilitate forensic reconstruction of failures – and you may be doing that already in your standard application.

What Unikernel Systems are available?

There are a good selection of unikernel systems already in place, supporting a large number of languages, but the number of unikernel projects continues to grow. Some of the more popular unikernel systems include:

- MirageOS: One of the oldest unikernels, it uses the Ocaml language.
- HaLVM: Another early unikernel, it is built on Haskell.
- LING: This long-established project employs Erlang.
- ClickOS: Optimized for network appliances, this has bindings for C, C++, and Python.
- OSv: A slightly different unikernel, it is based on Java with additional bindings which cover a large number of languages. Most JAR files reportedly drop in and run.
- Rumprun: Leveraging the modular code from the NetBSD project, it targets most any “POSIXy” application which doesn't need to fork. It is exceptionally well positioned for migrating existing applications into the unikernel world.

Are Unikernels a Panacea?

Unikernels are far from being a panacea. As they are single-process entities working in a single address space with no advanced memory management, there are certain programs which will not easily lend themselves to existence as a unikernel. However, a huge number of services running in datacenters across the world will easily fit the bill. By converting these services into lightweight unikernels, we can reassign the server capacity we freed up to the heavier tasks which could benefit from additional resources.

The number of tasks which lend themselves to being unikernels is larger than you might think. In 2015, Martin Lucina announced the successful creation of a “RAMP” stack. A variant of the common “LAMP” stack (Linux. Apache, MySQL, PHP/Python), the “RAMP” stack employs NGINX, MySQL, and PHP each built on Rumprun. Rumprun is an instance of a Rump kernel, which is a unikernel system based on the modular operating system functions found in the NetBSD project. So even this very common solution stack can be successfully converted into unikernels.

For More Information

To learn more about unikernels, check out http://www.unikernel.org/ or watch the talk I delivered at Southeast Linuxfest in 2015 at https://www.youtube.com/watch?v=8UgiPODw3CY .

About the Author:

Russell has spent over two decades evangelizing Open Source, serving most recently as the evangelist for the Xen Project. Since his introduction to Linux in 1995, he has relentlessly promoted the concept of Open Source to anyone who would listen.

He has over 200 pieces published, including columns for Infoworld and Processor magazines, one book published, plus one on the way. He has spoken over 100 times at Open Source conferences, including the biggest Linux conferences in North America. A former panelist on The Linux Show weekly webcast, he also has many years of experience employing Open Source software in solutions for clients.

He is currently looking for a new opportunity as an evangelist and/or community person for a FOSS project.

FREENAS MINI
STORAGE APPLIANCE

IT *SAVES* YOUR LIFE.

HOW IMPORTANT IS YOUR DATA?

Years of family photos. Your entire music and movie collection. Office documents you've put hours of work into. Backups for every computer you own. We ask again, *how important is your data?*

NOW IMAGINE LOSING IT ALL

Losing one bit - that's all it takes. One single bit, and your file is gone.

The worst part? **You won't know until you absolutely need that file again.**

Example of one-bit corruption

THE SOLUTION

The FreeNAS Mini has emerged as the clear choice to save your digital life. **No other NAS in its class offers ECC (error correcting code) memory and ZFS bitrot protection to ensure data always reaches disk without corruption and *never degrades over time*.**

No other NAS combines the inherent data integrity and security of the ZFS filesystem with fast on-disk encryption. No other NAS provides comparable power and flexibility. The FreeNAS Mini is, hands-down, the best home and small office storage appliance you can buy on the market. **When it comes to saving your important data, there simply is no other solution.**

The Mini boasts these state-of-the-art features:

- 8-core 2.4GHz Intel® Atom™ processor
- Up to 16TB of storage capacity
- 16GB of ECC memory (with the option to upgrade to 32GB)
- 2 x 1 Gigabit network controllers
- Remote management port (IPMI)
- Tool-less design; hot swappable drive trays
- FreeNAS installed and configured

iXsystems

http://www.iXsystems.com/mini

intel inside ATOM

FREENAS CERTIFIED STORAGE

With over six million downloads, FreeNAS is undisputedly *the* most popular storage operating system in the world.

Sure, you could build your own FreeNAS system: research every hardware option, order all the parts, wait for everything to ship and arrive, vent at customer service because it *hasn't*, and finally build it yourself while hoping everything fits - only to install the software and discover that the system you spent *days* agonizing over **isn't even compatible**. Or...

MAKE IT EASY ON YOURSELF

As the sponsors and lead developers of the FreeNAS project, iXsystems has combined over 20 years of hardware experience with our FreeNAS expertise to bring you FreeNAS Certified Storage. **We make it easy to enjoy all the benefits of FreeNAS without the headache of building, setting up, configuring, and supporting it yourself.** As one of the leaders in the storage industry, you know that you're getting the best combination of hardware designed for optimal performance with FreeNAS.

Every FreeNAS server we ship is...

» Custom built and optimized for your use case
» Installed, configured, tested, and guaranteed to work out of the box
» Supported by the Silicon Valley team that designed and built it
» Backed by a 3 years parts and labor limited warranty

As one of the leaders in the storage industry, you know that you're getting the best combination of hardware designed for optimal performance with FreeNAS. **Contact us today for a FREE Risk Elimination Consultation with one of our FreeNAS experts.** Remember, every purchase directly supports the FreeNAS project so we can continue adding features and improvements to the software for years to come. **And really - why would you buy a FreeNAS server from *anyone* else?**

FreeNAS 1U
- Intel® Xeon® Processor E3-1200v2 Family
- Up to 16TB of storage capacity
- 16GB ECC memory (upgradable to 32GB)
- 2 x 10/100/1000 Gigabit Ethernet controllers
- Redundant power supply

FreeNAS 2U
- 2x Intel® Xeon® Processors E5-2600v2 Family
- Up to 48TB of storage capacity
- 32GB ECC memory (upgradable to 128GB)
- 4 x 1GbE Network interface (Onboard) - (Upgradable to 2 x 10 Gigabit Interface)
- Redundant Power Supply

http://www.iXsystems.com/storage/freenas-certified-storage/

Kernel and syscalls - Introduction

by David Carlier

In this article, we will have an overview of what is called a syscall (system call shortened), from the kernel side to the userland, then in the end, how to create a new one in FreeBSD. It is assumed you know how to build FreeBSD currently and have some knowledge of C language.

What is a syscall?

A syscall is a code implemented in the kernel side which can be potentially called from the userland for various purposes; network, generating some random data, or controlling the system processes. Whatever the type of syscall, the userland never interacts directly with the kernel, rather via a defined interface. Let us start with a simple example, the getpagesize call which is the number of bytes per page.

```
int getpagesize(void);
```

Let us see how it is implemented in the kernel.

```
sys/kern/kern_mib.c

...

SYSCTL_INT(_hw, HW_PAGESIZE, pagesize, CTLFLAG_RD|CTLFLAG_CAPRD,

    SYSCTL_NULL_INT_PTR, PAGE_SIZE, "System memory page size");

...
```

Which simply returns the constant via the sysctl system. Let us see now how it is implemented in the userland's side:

```
lib/libc/gen/getpagesize.c
...
int
getpagesize(void)
{
        int mib[2];
        static int value;
        size_t size;
        int error;
        if (value != 0)
                return (value);
    /* Here we try to get the cached result as possible */
        error = _elf_aux_info(AT_PAGESZ, &value, sizeof(value));
        if (error == 0 && value != 0)
                return (value);
    /* Otherwise we get the result via the sysctl call */
        mib[0] = CTL_HW;
        mib[1] = HW_PAGESIZE;
        size = sizeof value;
        if (sysctl(mib, 2, &value, &size, NULL, 0) == -1)
                return (-1);
```

```
     return (value);

}

...
```

As you can see in this case, the work is mainly done in the userland side. It performs another type of syscall, where all the implementation is done in the kernel side only but its symbol is transmitted via a symbol map. Next, let’s have a look at the getpid syscall. The usual C function has this signature:

```
pid_t

getpid(void);
```

Those syscalls are organized per type in the kernel, and implemented in the files src/sys/kern/(kern/sys)_*.c. Basically, the sys_* files hold the function implementations, but if the complexity requires it, some codes are split in the kern_* ones.

First of all, the struct getpid_args reflects the userland parameters, every syscall needs a corresponding *args struct to pass the userland parameters, hence as getpid has none, here we have a dummy's.

```
src/sys/sys/sysproto.h

...

struct getpid_args {

        register_t dummy;

};

...
```

This type of syscall has this signature:

```
int <syscall>(struct thread *, struct <userland call's name>_args *)
```

The thread parameter is the current thread from which we can get the current process, locking it, in order to modify or just read a specific state...the integer return serves here to, potentially, set errno in errors code paths.

Here is the corresponding getpid implementation. In the case of getpid, it is always successful, errno is never set so we return 0 directly.

```
src/sys/kern/sys_prot.c

...

int

sys_getpid(struct thread *td, struct getpid_args *uap)

{

        struct proc *p = td->td_proc;


        td->td_retval[0] = p->p_pid;

#if defined(COMPAT_43)

        td->td_retval[1] = kern_getppid(td);

#endif

        return (0);

}

...

int

kern_getppid(struct thread *td)

{

        struct proc *p = td->td_proc;

        struct proc *pp;

        int ppid;
        PROC_LOCK(p);
        if (!(p->p_flag & P_TRACED)) {
ppid = p->p_pptr->p_pid;
                PROC_UNLOCK(p);
        } else {
                PROC_UNLOCK(p);
                sx_slock(&proctree_lock);
                pp = proc_realparent(p);
                ppid = pp->p_pid;
                sx_sunlock(&proctree_lock);
        }
        return (ppid);
}
...
/* Returns the real parent process whether it had exited or
traverses the orphaned linked list */
struct proc *
proc_realparent(struct proc *child)
{
struct proc *p, *parent;
 sx_assert(&proctree_lock, SX_LOCKED);
        if ((child->p_treeflag & P_TREE_ORPHANED) == 0) {
                if (child->p_oppid == 0 ||
                    child->p_pptr->p_pid == child->p_oppid)
                        parent = child->p_pptr;
                else
                        parent = initproc;
                return (parent);
        }
        for (p = child; (p->p_treeflag & P_TREE_FIRST_ORPHAN) == 0;)
{
                /* Cannot use LIST_PREV(), since the list head is not
known. */
                p = __containerof(p->p_orphan.le_prev, struct proc,
                    p_orphan.le_next);
                KASSERT((p->p_treeflag & P_TREE_ORPHANED) != 0,
                    ("missing P_ORPHAN %p", p));
        }
        parent = __containerof(p->p_orphan.le_prev, struct proc,
            p_orphans.lh_first);
        return (parent);
}
...
```

Now, the next step to see is how the kernel makes sys_getpid available to the userland.

Our getpid syscall can be found in sys/kern/syscalls.master file:

```
20      AUE_GETPID      STD     { pid_t getpid(void); }
```

20 is the syscall identifier, AUE_GETPID is the auditing event. Auditing in FreeBSD is the possibility to log different types of information from those syscalls: authentication, file descriptor operations (via fcntl call), userid/groupid changes, etc. In our case, in fact, AUE_GETPID is equal to AUE_NULL, which means no auditing. STD means it is a standard syscall.

Syscall function

Syscall was introduced in 4.0 BSD (circa 1980) and its signature is as follows:

```
int

syscall(int <syscall identifier>,...);
```

When it was introduced, there was no issue; due to having 32 and 64 bits types, it is not possible to use syscall function but `__syscall:`

```
but __syscall

off_t

__syscall(quad_t <syscall identifier>, ...);
```

Although it is practical to test a system call not available in the userland side, it is somehow not possible to have a function which returns values as arrays.

How to add your own syscall

FreeBSD allows adding new syscalls similar to the ones above. For example, let us make a syscall which returns the full command of a given pid (and if the pid is -1 then it would be the current one) which would have this signature:

```
int custom_func(pid_t pid, char *buf, size_t buflen);
```

It might be advised to create a new file instead of updating sys_generic.c, let us create sys_custom.c inside sys/kern folder:

```
 struct proc *p;

        int error = 0;
```

It might be advised to create a new file instead of updating sys_generic.c, let us create sys_custom.c inside sys/kern folder:

```
#include <sys/cdefs.h>

__FBSDID("$FreeBSD$");


#include <sys/param.h>

#include <sys/systm.h>

#include <sys/lock.h>

#include <sys/sx.h>

#include <sys/mutex.h>

#include <sys/proc.h>

#include <sys/errno.h>

#include <sys/sysctl.h>

#include <sys/sysproto.h>
```

// Here this is just for the example, having a “optin” to enable our custom function:

```
static int enable_custom = 0;

SYSCTL_INT(_debug, OID_AUTO, enable_custom, CTLFLAG_RW,

        &enable_custom, 0, "Enable custom");


int

sys_custom_func(struct thread *td, struct custom_func_args *uap)

{
```

```
td->td_retval[0] = 0;

        memset(uap->buf, 0, uap->buflen);


        if (enable_custom > 0) {

          // We set errno f the buffer is too small or

                if (uap->buflen < MAXCOMLEN) {

                        td->td_retval[0] = -1;

                        error = EINVAL;

                        goto out;

                }

// if we do not find our specific process

                if (uap->pid > -1) {

                        if ((p = pfind(uap->pid)) == NULL) {

                                td->td_retval[0] = -2;

                                error = ESRCH;

                                goto out;

                        }

                        PROC_UNLOCK(p);

                } else

                        p = td->td_proc;

// Before copying to the buffer the command, we lock the process

                PROC_LOCK(p);

                strlcpy(uap->buf, p->p_comm, MAXCOMLEN);
                PROC_UNLOCK(p);

        }

out:

        return (error);

}

...
```

To declare our new function, we need to add this in sys/kern/syscalls.master file:

```
...

550     AUE_NULL        STD     { int custom_func(pid_t pid, char
*buf, size_t buflen); }
```

Note the identifier must be unique, we just increment from the previous. Then in our FreeBSD source folder, we could type this:

```
make -C sys/kern sysent
```

Now there are new entries in sys/sys/sysproto.h header, first we have our userland struct arg:

```
struct custom_func_args {

        char pid_l_[PADL_(pid_t)]; pid_t pid; char pid_r_[PA-
DR_(pid_t)];

        char buf_l_[PADL_(char *)]; char * buf; char buf_r_[PA-
DR_(char *)];

        char buflen_l_[PADL_(size_t)]; size_t buflen; char bu-
flen_r_[PADR_(size_t)];

};
```

And our kernel function declared:

```
int      sys_custom_func(struct thread *, struct custom_func_args *);
```

We need to declare our new file sys_custom.c to the sys/conf/files file as below:

```
...
kern/sys_custom.c                standard
...
```

So the next time we compile the kernel it will be taken into account. Last, we need to make this function available, exporting its symbols:

```
lib/libc/sys/Symbol.map


FBSD_1.4 {
...
    custom_func;
...
}


FBSDprivate_1.0 {
...
    _custom_func;
              __sys_custom_func;
...
}
```

Once we have updated our system, our new function and our new sysctl ought to be available. We could test this new function with this short sample code:

```
int main(int argc, char **argv) {

...

char buf[MAXCOMMLEN];

pid_t pid = getpid();

if (custom_func(pid, buf, sizeof(buf)) == 0)

    printf("command of pid %d is '%s'\n", pid, buf);

}

...

% sysctl debug.enable_custom

0

./a.out

<no output>

% sysctl debug.enable_custom=1

0 -> 1

./a.out

command of pid 752 is 'a.out'

FBSDprivate_1.0 {

...

    _custom_func;

              __sys_custom_func;

...

}
```

Monitoring your syscall

To have an idea which calls are used during our tests, we can use the famous ktrace:

```
% ktrace -tc ./a.out (to trace only system calls)

% kdump

...

   979 a         CALL  custom_func(0xffffffff,0x7fffffffeac0,0x40)

   979 a         RET   custom_func 0

...
```

Apparently our syscalls went well, the return value is 0 with a valid pid, but with an invalid:

```
...

  1040 a         CALL  custom_func(0x4c4b41,0x7fffffffeab0,0x40)

  1040 a         RET   custom_func -1 errno 3 No such process

...
```

Commiting?

If you think a new syscall is needed, then you can either discuss it in the dev freebsd mailing list or propose a patch maintaining it locally; this kind of code change might bring difficulty when updating the source with new official syscalls added and conflicting identifiers. At least hopefully, this article gave you the desire to dig into this topic.

FreeBSD Kernel Module 1

by David Carlier

In this module, we will give an overview of the nature of the FreeBSD's kernel. The important configuration files will be explained in addition to learning how to compile the whole system with more options, with more debugging information. Very useful for kernel development.

Requirements:

- FreeBSD 10.x.
- Machine with at least 4 cores is recommended for the system compilation.
- Genuine hardware or virtualized environment as your convenience.

The FreeBSD kernel.

FreeBSD, like many kernels, is a monolithic kernel with loadable module support. Hence, it is possible to build FreeBSD's kernel with all needed modules statically or those ones that support it, as separated dynamic loadable modules.

The latter ones can be loaded and unloaded at will at boot time:

```
(<name of kernel module>_load="YES" in /boot/loader.conf file)
```

or via

```
kldload/kldunload.
```

To have an overview of all currently loaded modules, you can type kldstat. For example, the out-

```
Id Refs Address Size Name

1 19 0xffffffff80200000 19ff378 kernel

2 1 0xffffffff81e11000 4f62 ng_ubt.ko

…

9 1 0xffffffff81e5b000 3bab ng_socket.ko
```

put looks like the following:

Let's load the DTrace module by typing `kldload dtraceall.` Then if we type kldstat again, we should see some new entries related to this module:

```
…

10 1 0xffffffff81e5f000 89e dtraceall.ko

11 11 0xffffffff81e60000 9964 opensolaris.ko

12 10 0xffffffff81e6a000 857dba dtrace.ko

...
```

If we add `dtraceall_load="YES"`, we will be able to use Dtrace framework facility. You can find an excellent introduction of dtrace in the BSDmag issue of last December. Indeed, Dtrace can be very useful for tracing syscalls.

Configuration

In order to build the system, we need the whole source code, hence the kernel and the userland. The userland is simply all the base utilities of FreeBSD. Both kernel and userland code are consistent and tied together, available in the same subversion repository. Apart from pure BSD codes, we can find GNU libraries and software (called contrib code). In addition, for ZFS, CTF (Compact C Type Format debug section, similar to DWARF format but reduced in term of size) and DTrace proper compilations, some CDDL codes are present. Happily, FreeBSD is provided with subversion in base, suffixed distinctly to avoid colliding with the port version:

• checkout the source in /usr/src via svnlite

• svnlite co https://svn0.us-east.freebsd.org/base/stable/10 /usr /src (or you can check

out the current branch with much newer code but with more instability, you can just replace `stable/10 by head).`
To better understand how the kernel options work, let's have a look at `/usr/src/sys/conf/options` file:

```
...
# $FreeBSD$
#
# On the handling of kernel options
#
# All kernel options should be listed in NOTES, with suitable
# descriptions. Negative options (options that make some code not
# compile) should be commented out; LINT (generated from NOTES)
should
# compile as much code as possible. Try to structure option-using
# code so that a single option only switch code on, or only switch
# code off, to make it possible to have a full compile-test. If
# necessary, you can check for COMPILING_LINT to get maximum code
# coverage.
#
# All new options shall also be listed in either "conf/options" or
# this is a kernel-wide option (used just about everywhere), or in
# "opt_<option-name-in-lower-case>.h" if it affects only some files.
# Note that the effect of listing only an option without a
# header-file-name in conf/options (and cousins) is that the last
# convention is followed.
```

```
#
# This handling scheme is not yet fully implemented.
#
#
# Format of this file:
# Option name filename
#
# If filename is missing, the default is
# opt_<name-of-option-in-lower-case>.h
AAC_DEBUG opt_aac.h
AACRAID_DEBUG opt_aacraid.h
AHC_ALLOW_MEMIO opt_aic7xxx.h
AHC_TMODE_ENABLE opt_aic7xxx.h
AHC_DUMP_EEPROM opt_aic7xxx.h
AHC_DEBUG opt_aic7xxx.h
AHC_DEBUG_OPTS opt_aic7xxx.h
AHC_REG_PRETTY_PRINT opt_aic7xxx.h
AHD_DEBUG opt_aic79xx.h
AHD_DEBUG_OPTS opt_aic79xx.h
AHD_TMODE_ENABLE opt_aic79xx.h
AHD_REG_PRETTY_PRINT opt_aic79xx.h
ADW_ALLOW_MEMIO opt_adw.h
…
```

For each line, the option's name then the file created with the relevant preprocessor defined. If the option is present in your kernel configuration file, let's say `AHD_DEBUG_OPTS`, it is possible to test if AHD_DEBUG_OPTS is defined and providing some contextual code for this option. Let's imagine we did a new shiny kernel module, we could add our proper line in this file.

```
BSDMAG opt_bsdmag.h
```

Another important file is `/usr/src/sys/conf/file`.

```
...

cam/cam.c optional scbus

cam/cam_compat.c optional scbus

cam/cam_periph.c optional scbus

cam/cam_queue.c optional scbus

cam/cam_sim.c optional scbus

cam/cam_xpt.c optional scbus

cam/ata/ata_all.c optional scbus

cam/ata/ata_xpt.c optional scbus

cam/ata/ata_pmp.c optional scbus

cam/scsi/scsi_xpt.c optional scbus

cam/scsi/scsi_all.c optional scbus

cam/scsi/scsi_cd.c optional cd

cam/scsi/scsi_ch.c optional ch

cam/ata/ata_da.c optional ada | da

cam/ctl/ctl.c optional ctl

cam/ctl/ctl_backend.c optional ctl

cam/ctl/ctl_backend_block.c optional ctl
```

```
cam/ctl/ctl_backend_ramdisk.c optional ctl

cam/ctl/ctl_cmd_table.c optional ctl

cam/ctl/ctl_frontend.c optional ctl

cam/ctl/ctl_frontend_cam_sim.c optional ctl

cam/ctl/ctl_frontend_internal.c optional ctl

cam/ctl/ctl_frontend_iscsi.c optional ctl

cam/ctl/ctl_scsi_all.c optional ctl

cam/ctl/ctl_tpc.c optional ctl

cam/ctl/ctl_tpc_local.c optional ctl

cam/ctl/ctl_error.c optional ctl

cam/ctl/ctl_util.c optional ctl

cam/ctl/scsi_ctl.c optional ctl

cam/scsi/scsi_da.c optional da

cam/scsi/scsi_low.c optional ct | ncv | nsp | stg

cam/scsi/scsi_pass.c optional pass

cam/scsi/scsi_pt.c optional pt

cam/scsi/scsi_sa.c optional sa

...
```

For each line the relative path to sys, the type of module, if optional it will be compiled with the (lower case) option name written afterwards. Again, with our new module, we can add in our specific kernel module C file, let's say `workshop_module1.c`.

```
workshop_bsdmagmodule1.c optional bsdmag
```

Build

So now we can create a custom kernel config. Let’s call it WORKSHOP.

```
cp /usr/src/sys/<arch>/conf/GENERIC /usr/src/sys/<arch>/conf/WORKSHOP

echo "KERNCONF=WORKSHOP" >> /etc/make.conf
```

Steps to build a system:

First, the userland needs to be compiled.

```
/usr/src
```

If your machine has multiple cores, it is advised to use them for the system compilation. It might take several hours depending on your current configuration.

```
make -j<number of cores+1> buildworld (builds the userland)

make -j<number of cores+1> buildkernel (builds the kernel)

make installkernel (install the kernel in /)
```

You could possibly do (i.e., create and install the kernel at once):

```
make -j<number of cores+1> buildworld kernel
```

Restart in single user:

```
go to /usr/src
```

Eventually:

```
mergemaster -p
```

Then make:

```
installworld
```

And:

```
mergemaster -FUi
```

Mergemaster will try to merge various configurations files, asking you how you wish to proceed, merging as possible, replacing with a newer one or keeping the existing. Restart in normal mode.

You should have now a workable system with the latest fixes/patches for the 10.x branch. But, as a developer, we might need more info from the system for debugging, studying the core dump after a system crash/kernel panic. It is advised, as kernel developer, to enable kernel core dump writing (could be enabled when you installed FreeBSD or, afterwards, can be enabled via dumpdir rc.conf variable) at the cost of disk space consuming (can be potentially important, deleting old ones is necessary). They are, by default, located in /var/crash. In order to debug a kernel crash dump, the kernel compiled with debugging symbols, kernel.debug, is necessary. gdb can already be used this way.

kgdb

GNU gdb 6.1.1 [FreeBSD]

Copyright 2004 Free Software Foundation, Inc.

GDB is free software, covered by the GNU General Public License, and you are

welcome to change it and/or distribute copies of it under certain conditions.

Type "show copying" to see the conditions.

There is absolutely no warranty for GDB. Type "show warranty" for details.

This GDB was configured as "amd64-marcel-freebsd"...

Reading symbols from /boot/kernel/ng_ubt.ko.symbols...done.

Loaded symbols for /boot/kernel/ng_ubt.ko.symbols

Reading symbols from /boot/kernel/netgraph.ko.symbols...done.

Loaded symbols for /boot/kernel/netgraph.ko.symbols

Reading symbols from /boot/kernel/ng_hci.ko.symbols...done.

Loaded symbols for /boot/kernel/ng_hci.ko.symbols

Reading symbols from /boot/kernel/ng_bluetooth.ko.symbols...done.

```
Loaded symbols for /boot/kernel/ng_bluetooth.ko.symbols
Reading symbols from /boot/kernel/ng_l2cap.ko.symbols...done.
Loaded symbols for /boot/kernel/ng_l2cap.ko.symbols
Reading symbols from /boot/kernel/ng_btsocket.ko.symbols...done.
Loaded symbols for /boot/kernel/ng_btsocket.ko.symbols
Reading symbols from /boot/kernel/ng_socket.ko.symbols...done.
Loaded symbols for /boot/kernel/ng_socket.ko.symbols
Reading symbols from /boot/kernel/dtraceall.ko.symbols...done.
Loaded symbols for /boot/kernel/dtraceall.ko.symbols
Reading symbols from /boot/kernel/opensolaris.ko.symbols...done.
Loaded symbols for /boot/kernel/opensolaris.ko.symbols
Reading symbols from /boot/kernel/dtrace.ko.symbols...done.
Loaded symbols for /boot/kernel/dtrace.ko.symbols
Reading symbols from /boot/kernel/dtmalloc.ko.symbols...done.
Loaded symbols for /boot/kernel/dtmalloc.ko.symbols
Reading symbols from /boot/kernel/dtnfscl.ko.symbols...done.
Loaded symbols for /boot/kernel/dtnfscl.ko.symbols
Reading symbols from /boot/kernel/fbt.ko.symbols...done.
Loaded symbols for /boot/kernel/fbt.ko.symbols
Reading symbols from /boot/kernel/fasttrap.ko.symbols...done.
Loaded symbols for /boot/kernel/fasttrap.ko.symbols
Reading symbols from /boot/kernel/lockstat.ko.symbols...done.
Loaded symbols for /boot/kernel/lockstat.ko.symbols
Reading symbols from /boot/kernel/sdt.ko.symbols...done.
Loaded symbols for /boot/kernel/sdt.ko.symbols
```

Reading symbols from /boot/kernel/systrace.ko.symbols...done.

Loaded symbols for /boot/kernel/systrace.ko.symbols

Reading symbols from /boot/kernel/systrace_freebsd32.ko.symbols...done.

Loaded symbols for /boot/kernel/systrace_freebsd32.ko.symbols

Reading symbols from /boot/kernel/profile.ko.symbols...done.

Loaded symbols for /boot/kernel/profile.ko.symbols

```
#0 sched_switch (td=0xfffff8011b80a940, newtd=<value optimized out>,
flags=-2123250552) at /usr/src/sys/kern/sched_ule.c:1940

1940 cpuid = PCPU_GET(cpuid);
```

Like the userland gdb's counterpart, we can use backtrace (bt).

```
(kgdb) backtrace

#0 sched_switch (td=0xfffff8011b80a940, newtd=<value optimized out>,
flags=-2123250552) at /usr/src/sys/kern/sched_ule.c:1940

#1 0xffffffff8095b139 in mi_switch (flags=Unhandled dwarf expression
opcode 0x93

) at /usr/src/sys/kern/kern_synch.c:492

#2 0xffffffff8099b172 in sleepq_switch (wchan=<value optimized out>,
pri=<value optimized out>) at /usr/src/sys/kern/subr_sleepqueue.c:552

#3 0xffffffff8099afd3 in sleepq_wait (wchan=0xfffff80115316200, pri=U-
nhandled dwarf expression opcode 0x93

) at /usr/src/sys/kern/subr_sleepqueue.c:631

#4 0xffffffff8095aa47 in _sleep (ident=0x0, lock=0xfffff80115316230,
priority=0, wmesg=0xffffffff80ff47f2 "-", sbt=0, pr=0, flags=<value
optimized out>) at /usr/src/sys/kern/kern_synch.c:254
```

```
#5 0xffffffff8099f778 in taskqueue_thread_loop (arg=<value optimized
out>) at /usr/src/sys/kern/subr_taskqueue.c:118

#6 0xffffffff8091e234 in fork_exit (callout=0xffffffff8099f6b0
<taskqueue_thread_loop>, arg=0xfffff800078ebe90, fra-
me=0xfffffe0232e9fac0) at /usr/src/sys/kern/kern_fork.c:996

#7 0xffffffff80d4f4fe in fork_trampoline () at
/usr/src/sys/amd64/amd64/exception.S:610

#8 0x0000000000000000 in ?? ()
```

Also list if we want to see

```
(kgdb) list *0xffffffff8095aa47

0xffffffff8095aa47 is in _sleep (/usr/src/sys/kern/kern_synch.c:254).

249 else if (sbt != 0)

250 rval = sleepq_timedwait(ident, pri);

251 else if (catch)

252 rval = sleepq_wait_sig(ident, pri);

253 else {

254 sleepq_wait(ident, pri);

255 rval = 0;

256 }

257 #ifdef KTRACE

258 if (KTRPOINT(td, KTR_CSW))
```

Then, for example, going up in the stack frames calls and so on:

```
(kgdb) up 2

#4 0xffffffff8095aa57 in _sleep (ident=0x0, lock=0xfffff800035c6a30,

priority=0, wmesg=0xffffffff80ff47f2 "-", sbt=0, pr=0,
flags=<value optimized out>) at /usr/src/sys/kern/kern_synch.c:254
254 sleepq_wait(ident, pri);
(kgdb) list
249 else if (sbt != 0)
250 rval = sleepq_timedwait(ident, pri);
251 else if (catch)
252 rval = sleepq_wait_sig(ident, pri);
253 else {
254 sleepq_wait(ident, pri);
255 rval = 0;
256 }
257 #ifdef KTRACE
258 if (KTRPOINT(td, KTR_CSW))
```

For more information about gdb, a good helpful workshop exists about this topic:

http://bsdmag.org/course/application-debugging-and-troubleshooting-2

Indeed, especially if you run the -CURRENT branch, the kernel can crash for various reasons and this gdb like tool is handy to have a basic understanding of the reasons.

Detecting the potential deadlocks.

Indeed, FreeBSD does not rely on the Giant Lock model anymore, it is based on fine grained level process locking/unlocking. Hence the resulting programming can be tricky and it is easy to get lock contentions.

With this workshop module, you learned the basics of kernel custom configuration, compiling the whole system.

Exercise:

- In order to enable those additional capabilities, deadlock detections, more debugging info, additional checking … Which options in conf/options are needed?
- Recompile the kernel with those options.
- Once the system is restarted, which differences are you noticing? What are eventually the downsides?
- Is it possible to improve the situation? How?

About the Author:

David Carlier is a developer since 2001, mainly C/C++, living and working in Ireland mainly since 2012. He contributes to some open source projects and uses in a daily basis various op- erating systems mainly BSD flavours.

From Editor:

This is a first module of Developing FreeBSD Kernel Modules Online Course that used to be available on our web page. With Instructor we decided to take it down, because of incompleteness of course materials.

If you liked materials from this course, feel free to check our other online courses, which you can still join:

- **Application Debugging and troubleshooting**

https://bsdmag.org/course/application-debugging-and-troubleshooting-2/

- **Deploying on office / workgroup server on FreeBSD**

https://bsdmag.org/course/deploying-on-office-workgroup-servers-on-freebsd-2/

- **Practical Python Workshop**

https://bsdmag.org/course/python-programming-coming-next/

How to Upgrade OpenBSD and Build a Kernel

by Bob Cromwell

Let's see how to upgrade your OpenBSD system. Maybe you are doing this because the latest release just came out. If so, this is pretty simple: back up your data, boot from install media, and select "Upgrade" instead of "Install". But maybe the latest release has been out for a few months. Why would we go through the trouble of building and installing a new kernel or other core system components? Maybe some patches have been released to improve system security or stability. It is pretty easy to build and install a kernel on OpenBSD, easier and simpler in many ways than it is on Linux.

Verify that You Have the Real Operating System

If you purchase OpenBSD media then you have the real operating system. But what if you download it?

From the 5.5 release forward, you have the signify command along with the needed public keys in the `/etc/signify` directory. The OpenBSD project generates the key pair for the next release ahead of time, so you already have the public key for the next release. Once you have a trusted `/etc/signify` key collection, upgrade to every release and you can always verify things as you go.

This is better than the Linux kernel signing model! At the Linux kernel archive there is a digital signature file accompanying every kernel source code archive or patch file, but there is no clear mechanism for getting a trustworthy copy of the kernel organization's public key. The kernel.org site has some hand-waving about using a web of trust for PGP keys, then they tell you to track down a kernel developer in person and sign each others' keys. That's accompanied by a link to a Google map which absurdly tags a spot in the middle of the Atlantic Ocean as Cambridge, Massachusetts, one of the outer Aleutian Islands as Austin, Texas, and a spot in the White Sea off the Kola Peninsula as Oldenburg, Denmark. The most help I can provide you regarding the validity of the PGP signing key for the Linux kernel is that I'm convinced that it's key ID 0x6092693E with a fingerprint of:

```
647F 2865 4894 E3BD 4571 99BE 38DB BDC8 6092 693E
```

I'm convinced because that key has been used to sign the kernel source code for a number of years with no announcement of the site being hacked or the signing key being bogus.

But the good news with OpenBSD is that we have the security tools and keys in place. So, let's use them! Make sure you also download the file `SHA256.sig` accompanying the installation archives or source code archives.

How to use anonymous FTP

If you are preparing to patch a kernel or other parts of the core system (applications, shared libraries, etc), or if you want to build or update any of the ports, go to the appropriate release directory `/pub/OpenBSD/X.Y` and download files `sys.tar.gz` (the kernel), `src.tar.gz` (the rest of the core system), `ports.tar.gz` (ports), and also `SHA256.sig`, the digitally signed list of SHA-2-256 hashes of those archives.

If you are preparing an upgrade to a new release, go to the appropriate architecture-specific directory `/pub/OpenBSD/X.Y/arch` and download the appropriate bootable image `cdXY.iso` or installXY.iso, and also SHA256.sig, the digitally signed list of SHA-2-256 hashes of those image files. The file from the release directory `/pub/OpenBSD/X.Y` looks like this:

```
untrusted comment: signature from openbsd 5.8 base secret key

RWQNNZXtC/MqP81JgUxPz7/2d027cC49DG9Fq3gMJw52ZPlXy3pljMWTS74FmZ8lsQd2U
dPKzzVp2Qp/52NEsct0s5PTDO+gjAo=

SHA256 (ANNOUNCEMENT) =
1c4da371cf3138fafaac641b3c1030f67f4d73d5bcaa93318294007ae1a97304

SHA256 (HARDWARE) =
53eb55118bb57e2e94d3b9898819c959b8e7200537221cbee30c6797bac269f9

SHA256 (PACKAGES) =
cbc0964b2af61a8a8ec7b80174abd6d1a924663a03247b9687d76e5a65fef74e

SHA256 (PORTS) =
e8db12ce270e89289d71e6d6cc9e42b6f15d01ef305fe35bc2ec788f0b483bff

SHA256 (README) =
6d06b16db5b6fe3b0b33af42b53296512710bbbeebe53509f89968de2fac4d78

SHA256 (cd-src.tar.gz) =
74f1fe9a1ae3220ac6de30bf68e21a316f4bb5d7a465e8e2848977cf05f3b3fa

SHA256 (root.mail) =
0c89c7c9536760208c5ef33dd88cc9d2b1bd8270e26901177e7a39d9980c6c16

SHA256 (ports.tar.gz) =
9e840a89f84caca92be2b8dcbd4369ba1f96a1254473f59ed22da25649f4a9c1

SHA256 (xenocara.tar.gz) =
909244ce6aaa1a088b32dab77079c0a510856e299965e5e39f320972f53732bf

SHA256 (src.tar.gz) =
f3aee386ca66479d81a100860d64eb357a68a8c26b1e6edfbe184abd66954985

SHA256 (sys.tar.gz) =
78a4b191af0cd3121691fae05619e3a89938df1e89ff913d3c54aea0af490643
```

Now we can verify that we got the real software from the real OpenBSD project:

```
$ signify -C -p /etc/signify/openbsd-XY-base.pub -x SHA256.sig
```

The way that this works is that the second line of the `SHA256.sig` file is a signature for the file itself. If there has been any change (or addition or deletion) to the file, signify exits with the message "signature verification failed."

If the `SHA256.sig` file is good, you see "Signature Verified" and then one line for each of the listed files with its name and then "OK" or "FAIL". Here's the result of downloading just three archive files and using both a good signature file and one in which I changed one character of the hash for a file that I hadn't even downloaded:

```
$ signify -C -p /etc/signify/openbsd-XY-base.pub -x SHA256.sig

Signature Verified

ports.tar.gz: OK

src.tar.gz: OK

sys.tar.gz: OK

xenocara.tar.gz: FAIL

cd-src.tar.gz: FAIL

root.mail: FAIL

HARDWARE: FAIL

PORTS: FAIL

README: FAIL

PACKAGES: FAIL

ANNOUNCEMENT: FAIL


$ diff SHA256.sig SHA256.sig-modified

9c9

< SHA256 (root.mail) =
0c89c7c9536760208c5ef33dd88cc9d2b1bd8270e26901177e7a39d9980c6c16

---

> SHA256 (root.mail) =
0c89c7c9536760208c5ef33dd88cc9d2b1bd8270e26901177e7a39d9980c6c17
```

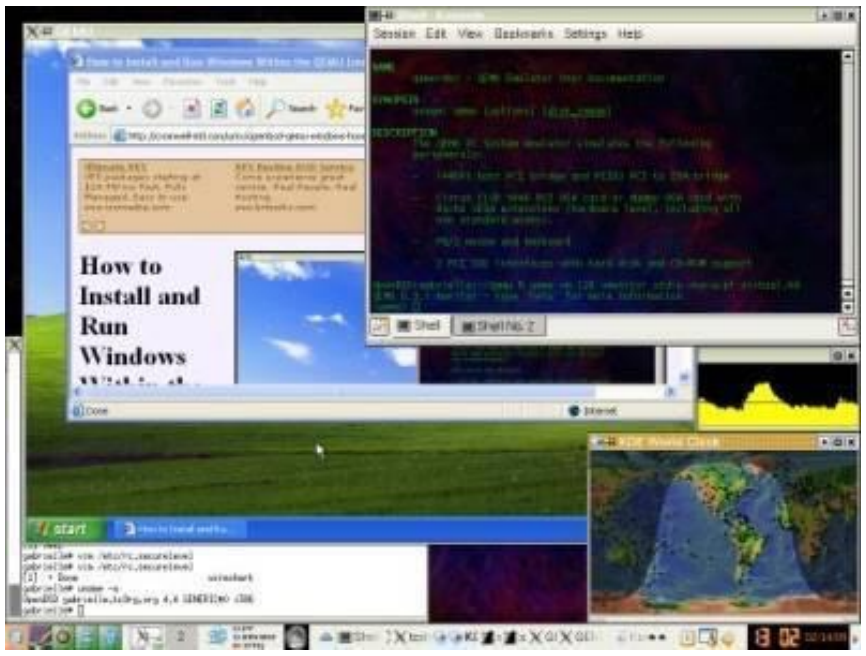
```
$ signify -C -p /etc/signify/openbsd-XY-base.pub -x
SHA256.sig-modified

signify: signature verification failed
```

Do not proceed until you see "OK" reported for each downloaded file. Of course, if you only downloaded some of the files listed in the SHA256.sig file, you will see "FAIL" for the missing files as in my example. That's fine.

Prepare for the Upgrade

Make sure your data is safely backed up before proceeding.



If you are starting a graphical display manager through `/etc/rc.local` or another mechanism, disable that temporarily.

Burn the appropriate OpenBSD installation media if you are upgrading to a new release. Boot from that media and select U for "Upgrade", and let it download and install the base operating system.

If you aren't building a new kernel, skip ahead to the steps needed to upgrade the packages. But if you are going to build a patched kernel…

Figure 1. Upgrading OpenBSD Kernel.

Install the Source Code

All the building and rebuilding on OpenBSD goes on under the /usr/src/ directory. It is empty when you first install the system. The source for the kernel and the userspace tools go into subdirectories.

Let's say that you downloaded the source code archives to /tmp. Install them:

```
# cd /usr/src

# tar xvfz /tmp/sys.tar.gz

# tar xvfz /tmp/src.tar.gz
```

Apply All Patches

Now you need to apply all the patches. They exist for good reasons — to fix problems of stability or security. Apply those patches!

The current patch list will be found at a URL resembling http://www.openbsd.org/errataXY.html, change XY to reflect your release minus the decimal point. Download the archive of all those patches, let's assume you also saved it to `/tmp/X.Y.tar.gz`. Extract the archive and become root:

```
$ cd /tmp

$ tar xvf X.Y.tar.gz

$ su

Password:

# cd X.Y/common

# ls -l
```

View each patch file to see its explanation of how to use signify and patch. Just copy and paste that two-line command into place. It will be something like this:

```
signify -Vep /etc/signify/openbsd-XY58-base.pub -x 011_sshd.patch.sig
\

    -m - | (cd /usr/src && patch -p0)
```

If the patch is for an application, the patch file comment block will tell you what to do next. Following that is the actual patch content. For example, building and installing the patched code will likely require something like this:

```
# cd /usr/src/usr.bin/ssh

# make obj

# make depend

# make

# make install
```

But if the patch is for the kernel, the patch file will just say "Then build and install a new kernel." Keep reading to see how!

Prepare to Build the Kernel

Change to the appropriate directory, changing `amd64` in the following commands as needed to reflect your hardware. OpenBSD runs on a lot of platforms, your choices include:

```
alpha, amd64, arm, armish, armv7, aviion, beagle, gumstix, hp300,
hppa, hppa64, i386, landisk, loongson, luna88k, m68k, m88k, mac68k,
macppc, mips64, mvme68k, mvme88k, mvmeppc, octeon, powerpc, sgi, sh,
socppc, solbourne, sparc, sparc64, vax, zaurus.

# cd /usr/src/sys/arch/amd64/conf
```

Examine and Possibly Change the Kernel Configuration

The file `GENERIC` contains the kernel configuration. It's just a little over 600 lines long, versus the 7,700 or more lines in a recent Linux kernel configuration file.

There are options near the top for support of GPT (GUID Partition Tables, needed for disks larger than 2 TB) and Microsoft's NTFS file system. Simply comment out a line with "#" to disable that option or leave a driver out of the kernel. As an example, I once had to disable (or omit) the apm driver when I was running old OpenBSD on even older hardware.

If you want to change the kernel message colors from the generic white text on a blue background, then first add two lines to:

```
/usr/src/sys/arch/amd64/conf/GENERIC
```

I don't care for the default white-on-blue. It looks too much like a Windows crash screen. You can specify the foreground text color (FG) and background color (BG) of console kernel messages:

```
option WS_KERNEL_FG=WSCOL_BLACK

option WS_KERNEL_BG=WSCOL_BROWN
```

You don't have many choices. According to:

```
/usr/src/sys/dev/wscons/wsdisplayvar.h
```

the only colors are black, red, green, blue, brown (really a dark yellow), magenta, cyan, and white. For what it's worth, white on green is kind of hard to read. Black on brown is as close to Purdue gold and black as it gets.

Multibooting OpenBSD & Windows

If you are multibooting with Windows, realize that OpenBSD assumes that the hardware clock is set to UTC while Windows assumes it is set to the local time. You can set the hardware clock to local time and then specify the OpenBSD kernel's offset in minutes. I live in the US-EST time zone, 5 hours or 300 minutes behind UTC, so I could set the hardware clock to US-EST and then do this:

```
option WS_KERNEL_FG=WSCOL_BLACK

option WS_KERNEL_BG=WSCOL_GREEN

option TIMEZONE=300
```

Virtualization commands:

However, I don't boot my laptop into Windows very often at all, especially now that I have a Windows virtual machine running under QEMU/KVM on my main Linux desktop. I just set the hardware to UTC, leave out the TIMEZONE option, and tell Windows that it's in UTC.

Build the Kernel

Configure the build with your `CONFIG` file:

```
# config GENERIC
```

Prepare to build, removing any leftover object files and setting up dependencies:

```
# cd /usr/src/sys/arch/amd64/compile/GENERIC

# make clean

# make depend
```

Build the kernel:

```
# make
```

That will take a few minutes, and should eventually result in a binary named bsd. On my system with an Intel Celeron 900 running at 2.2 GHz, it takes almost six minutes.

Install Your New kernel

This just takes one command, and watch what it does:

```
# make install

cmp -s bsd /bsd || ln -f /bsd /obsd

cp bsd /nbsd

mv /nbsd /bsd
```

If the new kernel file is different from the currently installed one, some files are shuffled around so that `/bsd` is the new kernel and `/obsd` is the previous one. That way, if some inappropriate changes in your kernel configuration file led to a dysfunctional kernel, you can reset the system and ask for the old one.

Reboot and Enjoy!

That should just simply work!

If it didn't, remember that you can ask for the old backup kernel at the boot prompt. And you don't have to remember its name, you can ask with the ls command to the boot prompt. And you don't even have to remember that, h gets you a list of the available boot prompt commands:

```
Using drive 0, partition 3.

Loading...

probing: pc0 apm mem[632K 3001M 780K 124K 36K a20=on]

disk: hd0+

>> OpenBSD/amd64 BOOT 3.28

boot> h

commands: # boot echo env help ls machine reboot set stty time

machine: boot comaddr diskinfo memory

boot> ls

commands: # boot echo env help ls machine reboot set stty time
```

```
machine: boot comaddr diskinfo memory
boot> ls
drwxr-xr-x 0,0  512      .
drwxr-xr-x 0,0  512      ..
drwxr-xr-x 0,0  512      home
drwxr-xr-x 0,0  512      tmp
drwxr-xr-x 0,0  512      usr
drwxr-xr-x 0,0  512      var
-rwxr-xr-x 0,0  9975220 bsd
-rw-r--r-- 0,0  7642500 bsd.rd
drwxr-xr-x 0,0  512      altroot
drwxr-xr-x 0,0  1024     bin
drwxr-xr-x 0,0  19456    dev
drwxr-xr-x 0,0  3584     etc
drwxr-xr-x 0,0  512      mnt
drwx------ 0,0  1024     root
drwxr-xr-x 0,0  1536     sbin
-rw-r--r-- 0,0  578      .cshrc
-rw-r--r-- 0,0  468      .profile
stat(hd0a:/./sys): No such file or directory
-rwxr-xr-x 0,0  9975220 obsd
-rw-r--r-- 0,0  69652    boot
boot> b /obsd
```

Running OpenBSD on old laptops

If you have the APM issue that required a kernel configuration change to install and boot in the first place, then see my page about installing OpenBSD on a Dell laptop to refresh your memory as to how to accomplish this.

The alternative would have been to modify the kernel configuration by editing the file:

```
/usr/src/sys/arch/amd64/conf/GENERIC
```

before building a kernel that leaves out that driver.

However, if you are really qualified to modify the kernel configuration before building and installing it, I'm not sure why you're reading this page!

I got lost!

Take me back to the kernel build!

Upgrading Packages

You will need to upgrade all the added packages. It is very easy to upgrade the packages you installed from the OpenBSD distribution:

```
# pkg_add -u
```

The core of the operating system, what you get with an installation, has been checked very carefully. The much larger collection of packages, basically those components that will be installed under the `/usr/local` hierarchy, haven't been checked as carefully. However, at least some design and implementation auditing is done and the packages are digitally signed. Things like the KDE Desktop Environment will be buggier than the plain X desktop provided in the core release, but at least you're getting just naturally occurring bugs and not Trojan Horses.

The `pkg_add` program checks digital signatures and will only install those with a good signature. You can see what's going on by downloading and extracting the gzip-compressed tar file for a package. You will get the collection of files making up the package plus +CONTENTS, +DESC, and possibly others named +*. The +CONTENTS file contains something like the following from the aescrypt package, which has the security parts highlighted:

```
@comment $OpenBSD: PLIST,v 1.2 2004/09/15 18:35:59 espie Exp $

@name aescrypt-0.7p0

@signer openbsd-58-pkg

@digital-signature
signify:2015-08-09T11:42:52Z:RWRlkI2aFHvL/YZnu9YhbPdOyWg8b73HvaFQiCgB
rbwWRyWrqYkzxocPvyzgIviAKAbN2o1D/a1UYW6a4vfcgSAHCBGeu8tOYQo=

@comment pkgpath=security/aescrypt cdrom=yes ftp=yes

@arch amd64

+DESC

@sha Sise4CvlTcCZ3vOao4uR26nPXM31XIzXi80Z2M1XeuA=

@size 416

@wantlib c.80.1

@cwd /usr/local

bin/aescrypt

@sha fnHWBdfHf7WLMR33Mk91jxQ/3yqyoX4j/ixJ4ZNEr5c=

@size 36608

@ts 1438963247

bin/aesget

@sha XARARn2qi0R6udvoXJT6Sx/O5Js4Nx/e1/Zfv80p6ws=

@size 36608

@ts 1438963247

share/doc/aescrypt/

share/doc/aescrypt/README.html

@sha CKtEtiNFdKjbZSi3xyaXoT2Btz2yPlFbDmkTkhApDf8=
@size 4792

@ts 1438963247
```

The `@digital-signature` line contains a digital signature for the `+CONTENTS` file itself. That digital signature must be valid in order to continue. Now we can trust the `@sha` lines to tell us the `Base64-encoded SHA-2-256` hashes of the components `+DESC`, `bin/aescrypt`, `bin/aesget`, and `share/doc/aescrypt/README.html`. This is pretty well hidden from the administrator, you only see hints of what's going on when you `pass -vvv to pkg_add` to tell it to be triply verbose! Let's verify what's happened once the package is installed:

```
# sum -a sha256 -b /usr/local/bin/aescrypt /usr/local/bin/aesget
/usr/local/share/doc/aescrypt/README.html

SHA256 (/usr/local/bin/aescrypt) = fnHWBdfHf7WLMR33Mk91jxQ/3yqyoX4j/
ixJ4ZNEr5c=

SHA256 (/usr/local/bin/aesget) = XARARn2qi0R6udvoXJT6Sx/O5Js4Nx/e1/
Zfv80p6ws=

SHA256 (/usr/local/share/doc/aescrypt/README.html) = CKtEt-
iNFdKjbZSi3xyaXoT2Btz2yPlFbDmkTkhApDf8=

  -- or, alternatively --

# for F in /usr/local/bin/aescrypt /usr/local/bin/aesget
/usr/local/share/doc/aescrypt/README.html

> do

>   echo $F $( sopenssl sha256 -binary $F | base64 )

> done

/usr/local/bin/aescrypt fnHWBdfHf7WLMR33Mk91jxQ/3yqyoX4j/ixJ4ZNEr5c=

/usr/local/bin/aesget XARARn2qi0R6udvoXJT6Sx/O5Js4Nx/e1/Zfv80p6ws=

/usr/local/share/doc/aescrypt/README.html CKtEtiNFdKjbZSi3xyaX-
oT2Btz2yPlFbDmkTkhApDf8=
```

Notice that the measured SHA-2-256 hashes agree with what was specified in the +CONTENTS file — the pkg_add program wouldn't have installed the package if they hadn't matched those in that digitally-signed list!

Reconfiguring Packages

If you use the KDE display manager kdm, then you may have fixed it to start an SSH agent at login as described here. You will need to do that again if you want to continue using that method. However, maintenance is easier if you use Keychain instead.

If you are using LaTeX and/or TeTeX in a region of the world with a mutant paper size (like the U.S. and Canada), then you need to run texconfig as root and specify your paper size.

If you are using Wireshark or the command-line tshark, remember to change the devices /dev/bpf* to be readable by a group containing those users who should be able to capture packets in that group. The details are here.

Rebuilding Other System Binaries

The above recommendation on downloading, installing, and patching the application source code will have some of the work out of the way. At least you will have patched and trusted source code.

What remains is to build and install the binaries based on the patched code. See the patch files for the details, but this generally requires something like the following, where you would skip the patch application if you have already done it:

```
# cd /usr/src/path/to/patched/source/code

# make

# make install
```

If this fails with an error message message about "don't know how to make ...", then you need to clean things up before doing the build. Do the following and then restart building that binary. Notice that the pushd and popd leave you wherever you were before applying this fix:

```
# rm -rf /usr/obj/*

# pushd /usr/src

# make obj

# popd
```

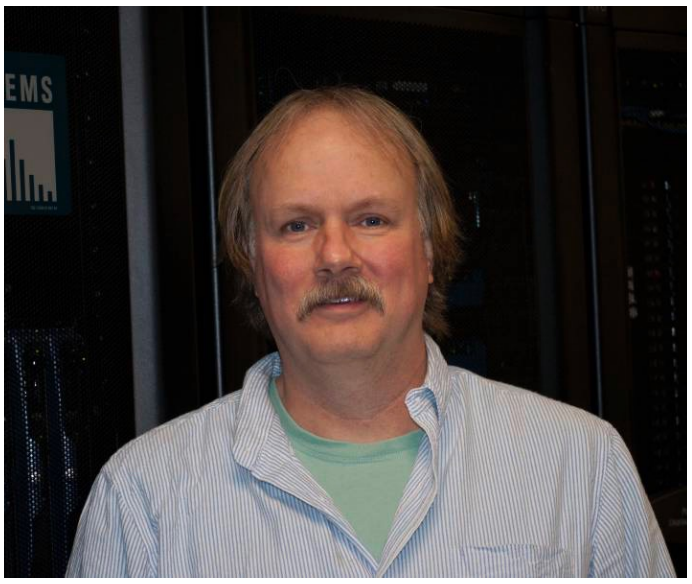
Rebuilding Other Open-Source Packages

If you built and installed any open-source packages under the old version, you will need to rebuild them to match the current set of shared libraries (and this may involve rebuilding some shared libraries you built and installed). See my pages on building Wireshark, OpenCV, OpenVAS, and RainbowCrack on OpenBSD.

How to Compile Software on OpenBSD:

Wireshark OpenCV OpenVAS Rainbow Crack

How to build Linux kernels How to install and run OpenBSD on a low-end Dell laptop Various Linux / UNIX topics



About the Author:

I've been using OpenBSD since, well, not sure how long... Some time in the late 1990s. I've used Linux since you downloaded 40+ floppy images, some time around 1993-1994. Before that I had used UNIX, SunOS and forms of BSD, at Purdue since the mid 1980s. I got a BSEE at Purdue back then, worked at the university, grad school, Ph.D. in electrical and computer engineering, have done consulting since 1992. I've taught courses for Learning Tree International since the mid 1900s, and have written courses for them since the late 1990s.

Maxing Out Storage Performance with ZFS Caching

by Mark VonFange

One of the more beneficial features of the ZFS filesystem is the way it allows for tiered caching of data through the use of memory, read and write caches. By optimizing memory in conjunction with high speed SSD drives, significant performance gains can be achieved for your storage.

The first level of caching in ZFS is the Adaptive Replacement Cache (ARC), which is composed of your system's DRAM. It is the first destination for all data written to a ZFS pool, and it is the fastest (i.e. lowest-latency) source for data read from a ZFS pool. When data is requested from ZFS, it looks first to the ARC; if it is there, it can be retrieved extremely quickly (typically in nanoseconds) and provided back to the application.This provides greater read performance improvements - by orders of magnitude - over older methodologies like short-stroked spinning disks, which are power hungry and expensive.

The contents of the ARC are balanced between the most recently used (MRU) and most frequently used (MFU) data. This balance is important. Normally, running a backup product would walk the entire file system and effectively invalidate the cache. Since ZFS utilizes algorithms to track frequently used data in addition to recently used data, your cache devices will still provide performance improvements after a backup. ZFS brings frequently and recently used data to the highest performing storage, first to system memory, then to caching devices, allowing for flash media performance without the cost.

Level 2 Adaptive Replacement Cache (L2ARC)

Once all the space in the ARC is utilized, ZFS places the most recently and frequently used data into the Level 2 Adaptive Replacement Cache (L2ARC). The L2ARC is usually larger than the ARC so it caches much larger datasets. ZFS will accelerate random read performance on datasets far in excess of the size of the system main memory, which avoids reading from slower spinning disks as much as possible.

The ZFS Intent Log (ZIL)

ZFS commits synchronous writes to the ZFS Intent Log, or ZIL. Super-capacitor-backed DRAM as SLOG (Separate) ZIL devices allow cached writes to be committed to nonvolatile storage so that they are protected in the event of a sudden power failure. This allows synchronous writes to be made at the speed of the SLOG device, accelerating write performance.

Determining Cache and Pool Size

After the ARC, ZIL, and L2ARC comes the hard disks, comprising the ZFS pool. This tier is where your data lives and is usually composed of high capacity hard disks. Performance at this tier is the lowest of all, as it depends on spinning disks rather than flash drives.

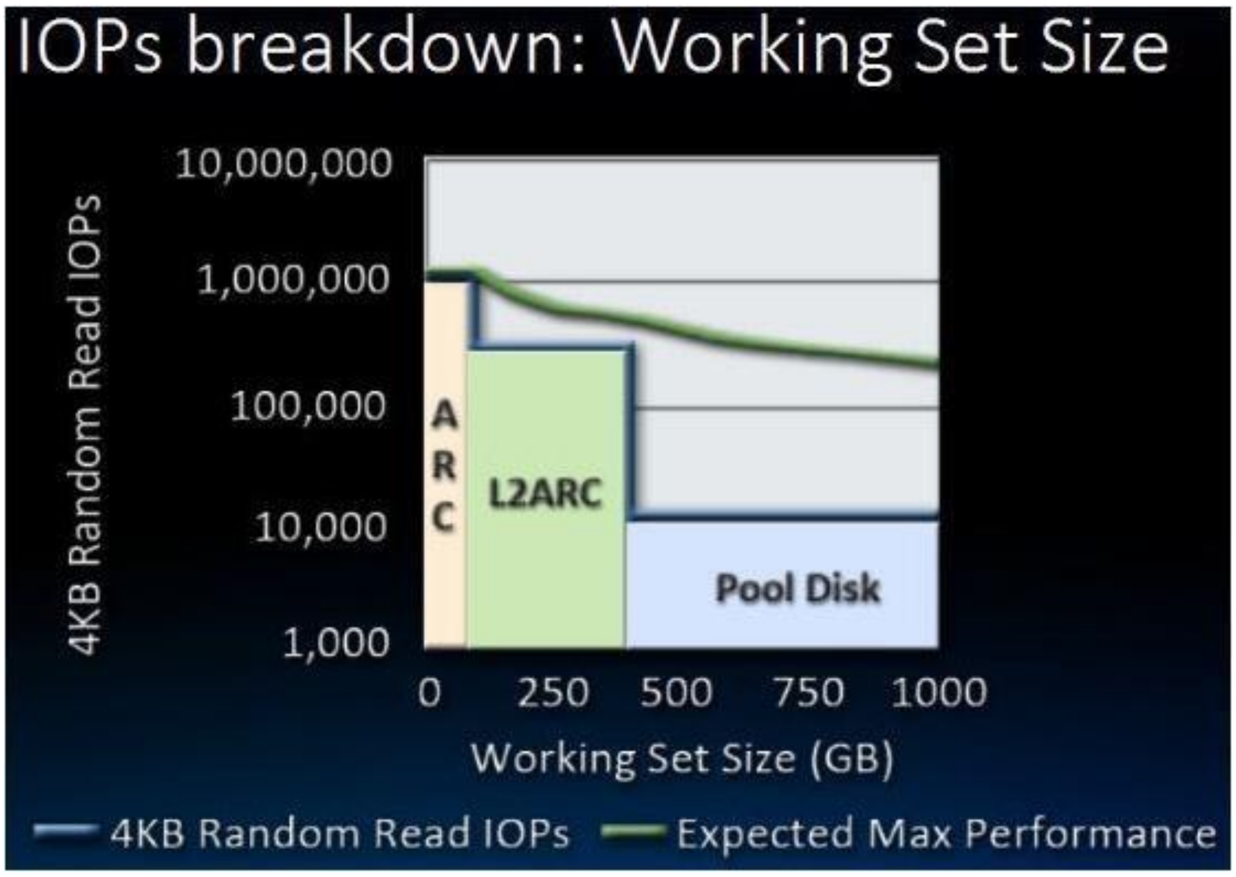


Figure 1. IOPs Breakdown: Working Set Size

IOPS by Type of Storage

In order to configure your FreeNAS or TrueNAS system for ideal performance between cache and pool, it is important to determine the Working Set Size of your system. Knowing the active data and performance requirements of your storage environment will allow you to put together a system that maximizes performance. Some questions that can help determine the Working Set Size are as follows:

- What percentage of your total data is "active?" (20% is not unusual)?
- How will the remaining data be dealt with?
- One file or a set of data accessed simultaneously?
- How many users or applications?
- How many people will log in simultaneously?
- What is the average file size and how many?
- What is your workflow?
- What percentage of your usage is read versus write?

Once the Working Set Size is determined, one can select the optimal drive to maximize performance. In regard to types of SSD cache drives, the L2ARC read cache does not require as high performance, as the data is already stored on disk and there is no risk of data loss. The ZIL requires higher quality storage devices/memory, as the data has not made it to the storage pool yet. In the event of a power loss, cheaper flash memory (MLC flash) can lose write data. SLC Flash Memory devices, on the other hand, do not have this issue.

Conclusion

ZFS Caching can be an excellent way to maximize your system performance and give you flash speed with spinning disk capacity and cost. FreeNAS and TrueNAS capitalize on this technology, allowing you to design a system that fits your needs and leverages the caching capabilities of ZFS to their full extent.

Mark VonFange

iXsystems™

Model View Whatever - Dolphin Smalltalk MVP

by Damian Czernous

The Dolphin Smalltalk is a version of Smalltalk language dedicated to the Windows platform. The Smalltalk language comes with a widget library done with the Model View Controller (MVC) structure. Dolphin, however, deviates from that path.

In 1995, dolphins at Object Arts start working on a Taligent version of the Model View Presenter described in the previous paper „Model View Whatever - MVP" by Mike Potel. In 2000, in San Diego, Andy Bower and Blair McGlashan (Object Arts) publish a paper „Twisting The Triad - The evolution of the Dolphin Smalltalk MVP application framework". Their roots are strong. In contrast to Mike Potel (Taligent version of the MVP), they pay more attention to the communication between the view and the presenter. This predilection is not accidental, and comes from the MVC experience. It also makes the whole MVP work more complete.

Taligent company

This very interesting company was founded in 1992 as a part of agreement between Apple Computers and IBM. The idea was to create an object-oriented operating system called Taligent that could run on every hardware platform. The history of the company is quite stormy. The system never came out. A year after the CEO's death, in 1996, Taligent becomes a subsidiary of IBM to be finally dissolved in 1998. Some refer to Taligent as an example of project death march.

During these six years, however, Taligent creates CommonPoint that could run on any modern operation system, e.g. AIX, HP-UX, OS/2, Windows NT, or Apple OS, providing features like: compound documents, 2D and 3D graphics, and real-time document sharing and collaboration. The company provides Open Class libraries for IBM's VisualAge, and Java WebRanner. They license key Java and C++ technologies to partners such as Sun, Netscape, and Oracle.

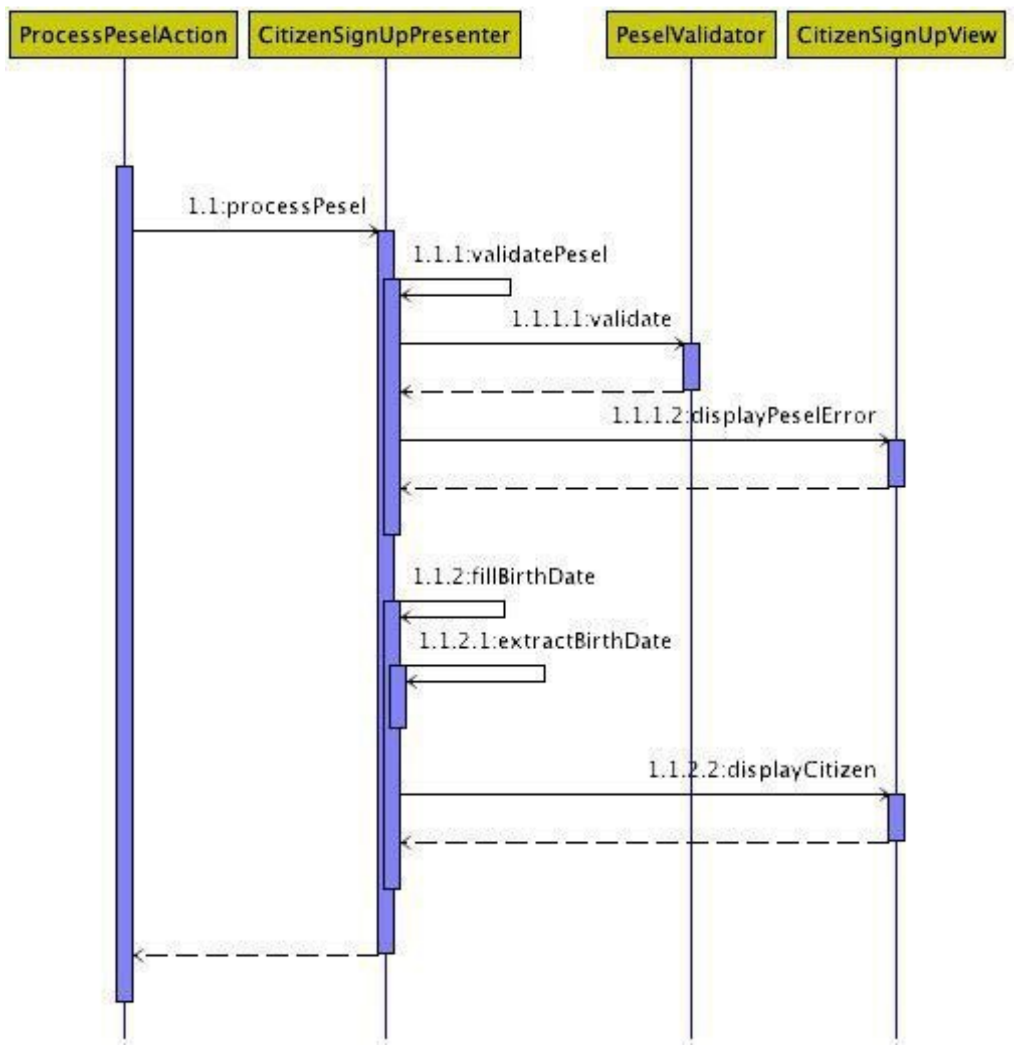
A number of Taligent classes live inside Sun's Java Development Kit (JDK). The work of engineers result in more than 300 patents. The company behaves more like a decent university with a number of pioneering projects that influence the work of many industry partners. Thank you, Taligent engineers.

Application Model lesson learned

VisualWorks Application Model enters the stage in the 90's. The MVC developed by Xerox Parc (by Trygve Reenskaug) seems to be interesting for ParcPlace engineers. They create a version of Smalltalk platform called VisualWorks that can run on many operating systems long before Java. For the GUI, they create a variation of Presentation Model (described in a paper "Model View Whatever - MVC's model evolution") called the Application Model.

The Presentation Model implements an idea of separating widget states and stylings from the application domain. The structure makes the original MVC more front end friendly. The Application Model introduces Property Objects that make widget to model mapping a bit easier.

The Property Objects allow widgets to observe the model using Observer Synchronisation mechanism. That requires additional and unwanted widget implementation. The result is that the Application Model manipulates widgets directly. For Dolphin, overcoming such edits in MVP becomes important. They look for a chance to discuss how the presenter should control the view.



Supervising Controller

The term was coined by Martin Fowler in his GUI Architecture work. As a result of the discussion, Bowler and McGlashan end up with the presenter that handles user actions and „more important" view logic. The „less important" cases, however, are administrated (using declarative paradigm - see the marked parts of the code below) by the presenter, and are implemented inside the view (Figure 1).

Figure 1: A controller that supervises a view.

More vs less important view logic

Validating an entered PESEL number and extracting a birth date to the age field (Figure 1) might be seen as a more important than displaying a hint for a field. The assumption is that a Polish user might be disappointed calculating manually the age or saving the wrong PESEL number, but may be fine without seeing a hint for the well known Polish national identification number (PESEL). The more important view logic is the one the user will not want to live without it.

This interpretation is not the one implied by Martin Fowler in his description, but I found it more practical. Martin talks more about complexity of an algorithm itself rather than its importance. Thus, the more complex view logic remains within the presenter. The simple view logic in the view.

```
public class ProcessPeselAction implements
Property.ValueChangeListener

{

private final CitizenSignUpPresenter citizenSignUpPresenter;

@Override

public void valueChange( Property.ValueChangeEvent valueChangeEvent )

{

citizenSignUpPresenter.processPesel();

}

}

public class CitizenSignUpPresenter

{

private final CitizenSignUpView citizenSignUpView; private PeselVali-
dator peselValidator;

public void processPesel()

{
```

```
if( Objects.nonNull( peselValidator ) ) validatePesel(); fillBirth-
Date();

}

public void fillBirthDate()

{

Citizen citizen = citizenSignUpView.getCitizen();
citizen.setBirthDate( extractBirthDate( citizen ) );
citizenSignUpView.displayCitizen( citizen );

}

private String extractBirthDate( Citizen citizen )

{

try

{

return Citizen.extractBirthDate( citizen.getPesel() );

}

catch( Citizen.DateFormatException e )

{

return "";

}

}

public void validatePesel()

{

try

{

peselValidator.validate( citizenSignUpView.getCitizen().getPesel() );
}

catch( PeselValidator.PeselValidationException peselValidationExcep-
tion )

{

citizenSignUpView.displayPeselError(
peselValidationException.getReason() );

}

}

}
```

View logic is not a business logic

Different UIs may have different view logics for the same business logic. The same application may have web UI, mobile UI, desktop UI, command line UI, printer interface, etc. Each may use business logic to get a job done. The business logic stays apart from any MV* structure. It does not belong to the front end part of the application. User authentication is an example of the business logic, and at most should be called by the controlling part when needed.

Potel's way and Passive View

Testability is an another way of thinking around Supervising Controller mechanism. Mike Potel does not say much about the view/presenter communication. That means that the whole view logic may stay inside the view. The view objects are usually difficult to test due to the number of bindings to the widget framework.

The Supervising Controller keeps essentials separated from the view thanks to the declarative communication. That makes the presenter's view logic fairly easy to test, and the less important parts most likely untested. The Passive View18 approach, however, moves the entire view logic to the controlling part making it almost fully testable. The cost of that is the complex controller that imperatively communicates with the view (see the code on the next page).

```
public class CitizenSignUpPresenter

{

private final CitizenSignUpView citizenSignUpView; private PeselVali-
dator peselValidator;

{

String pesel = citizenSignUpView.getPeselTextField().getValue();
String birthDate = extractBirthDate( pesel );
citizenSignUpView.getBirthDateTextField().setValue( birthDate );

}

public void validatePesel()

{

try

{

peselValidator.validate(
citizenSignUpView.getPeselTextField().getValue() );

}

catch( PeselValidator.PeselValidationException peselValidationExcep-
tion )

{

Notification.show( peselValidationException.getReason() );

}

}

}
```

In next paper

In the age of small modules, or thinking about software as a combination of small pieces, engineers notice that writing the controlling part is more about code plumbing than coding the meaningful view logic. The web applications are everywhere. In a short time, the controlling part will disappear, and the view logic will find a place much closer to the model than ever before. Have a look on Model View Whatever - view and two models

About the author:

Damian Czernous

Reasoning about software architecture fascinates

me for 10 years now.

Lead Engineering Coach at Nokia.

www.sanecoders.com, @DamianCzernous

References:

http://www.object-arts.com/downloads/papers/TwistingTheTriad.PDF

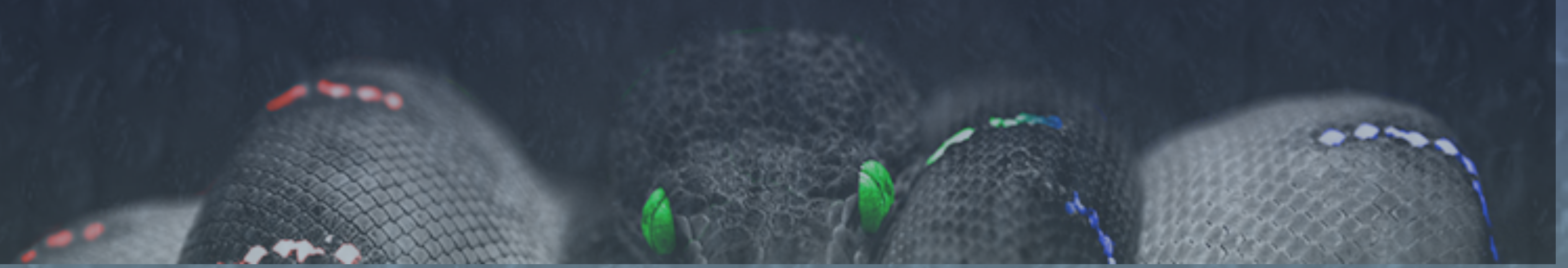
https://web.archive.org/web/19971211222028/http://www.taligent.com

http://martinfowler.com/eaaDev/PresentationModel.html

http://martinfowler.com/eaaDev/PassiveScreen.html

PRACTICAL PYTHON WORKSHOP

PEDRO ARAÚJO
& RUI SILVA



Module 1 (introduction):

Why python?

- Introduction about python programming lan- guage.
- Learning the strengths of the language and what's good with python.
- Learning where to use Python and why.
- Python as an interpreted language
- How to choose correct interpreter, install it, run it.
- Python virtual environments.
- Text editor (kate, gedit, brackets).
- How to create Hello world, from interpreter and with .py script.
- Standards and batteries included
- Standards and PEP8.
- Batteries included (just to show the most useful python core libraries and link to the official documentation).

•

Module 2 (python basics):

- Python data types and flow control state- ments
- Ifs, fors, whiles
- Lists (slices), dictionaries (loop over items),
sets
- Python internals
- Classes and object instances
- Everything is an object (docs strings, get- ters, setters, override)
- Exceptions handling
- Practical example
- Use twitter's API to get some data and show it in the console

Module 3 (files):

• Files
• Duck typing.
• Opening and reading from files. • Csv files and csvreader.
• Practical exercise

Read file with a sentence per line.
• Manipulate and gather metrics on each sen-

tence.
Output a file with metrics on each sentence.

Module 4 (project):

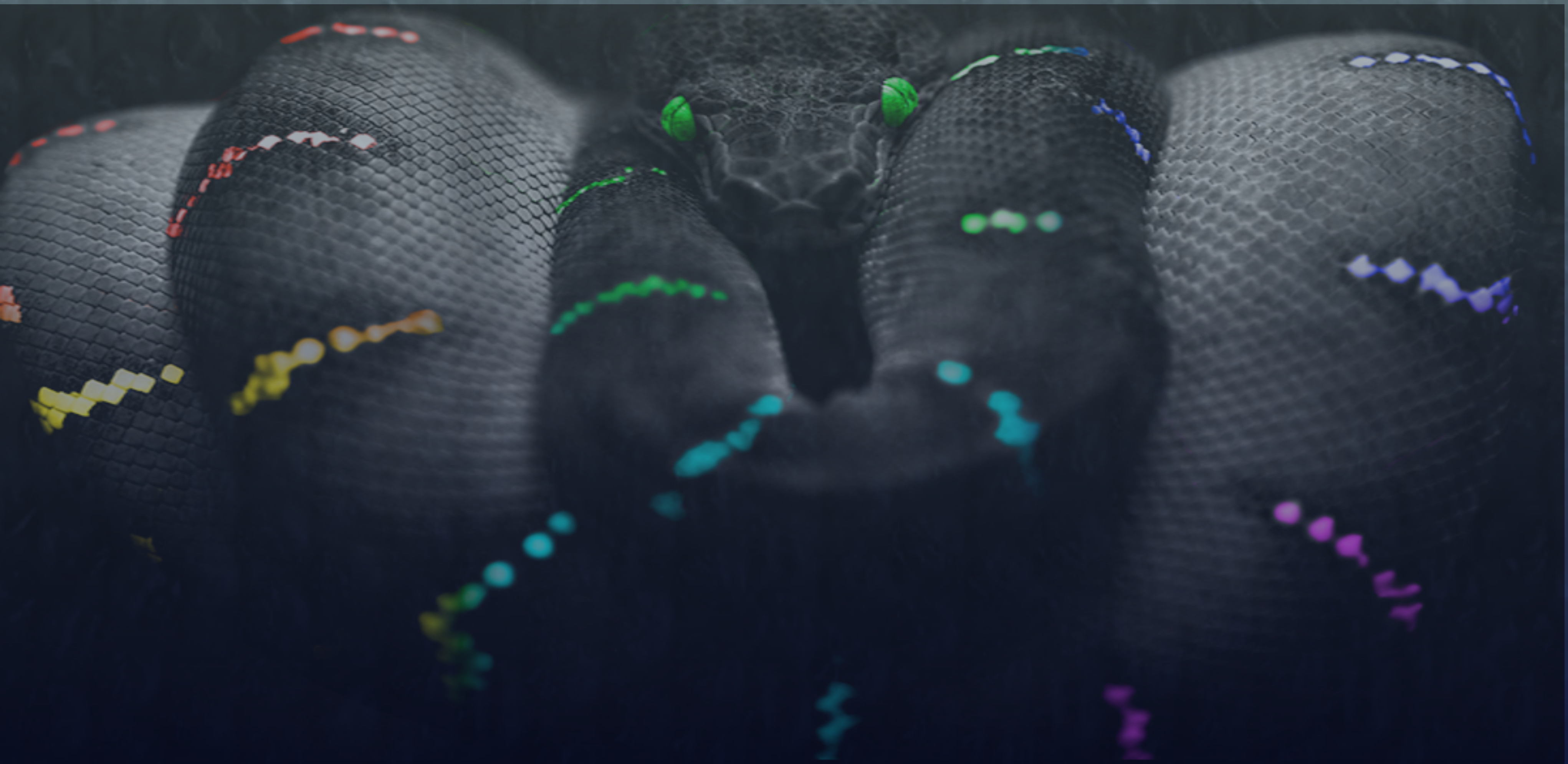
• Practical project
• Get data from external source (http://openweathermap.org).
• Manipulate data to suit our needs.
• Plot a graph to show the data in a graphical

and understandable way.

Authors

Pedro Araujo and Rui Silva

If you have any questions or just want to get to know us better feel free to contact me at marta.ziemianowicz@bsdmag.org or visit https://bsdmag.org/course/python-programming-coming-next/



Before you start, you have to have a passion that borders on obsession...

Giuseppe Canale from Escom Net

by Marta Ziemianowicz, Marta Strzelec & Marta Sienicka

[BSD Magazine]: Hello Giuseppe, how have you been doing? Can you introduce yourself to our readers?

[Giuseppe Canale]: Hello and thanks so much for having me! To give you a little background, I'm a small business owner in Northern Italy - specifically in Alba, which is about 45 minutes south of Turin, where the 2006 Winter Olympics were. I got into it at a time when, at least in Italy, IT hardly existed. My first experience on the net was with a Commodore 64 and a modem with a 300 baud rate (1 Bd = 1bit/s) and a ITAPAC (X.25 protocol) connection. In fact, when I went to college, I knew I wanted to major in IT, but it wasn't even possible - the course didn't exist! So I majored in mechanical engineering and did a lot of independent study outside of school.

After I graduated, I started off selling hardware - this was in 1987 and quickly thereafter, I got into software development. In '97 I opened up my own company. It was an exciting time in technology - e-commerce was just being invented…but Italy was and continues to be somewhat behind-the-times when it comes to IT.

[BSD Mag]: Do you know why Italy is behind when it comes to IT?

[GC]: Technology had two phases: pre and post internet. And for Italy, the post-internet phase is what created the problem. The Internet requires a connection and for a series of motives, principally geographical, but also logistical, Italy is difficult to physically connect. Besides the fact that we have lot of mountains, and hilltop villages and the majority of our towns are ancient/medieval, it's hard to find a town or village in Italy that isn't a historic center of some sort. So, digging to lay the lines is a problem - you either need to tunnel through massive quantities of rock, deviate around mountains or avoid destroying unrecoverable historic artifacts. Often, when they actually decide to attempt installing lines, they uncover some archeological ruin and all work stops indefinitely. Also, since the towns were designed so long before even electricity was developed, the cities weren't created with urbanization in mind. In many places, even installing running water and central sewage was a major problem. These are the first huge obstacles that we're still trying to overcome.

[BSD Mag]: Can you tell us something about the company you work for, Escom Net?

[GC]: I like to think of my company as a sort of laboratory rather than a traditional office. Our core focus has changed over the years to respond to changing technology (from hardware, to software, to mobile, e-commerce, social media, security…). Continuing education is very important to me and to my business partners - we're constantly doing research, reading the latest tech news and developing projects to try out some new strategy or put our new found skills to the test.

[BSD Mag]: What will be, in your opinion, the "next big thing" that we're all going to have to educate ourselves about?

[GC]: Protecting our digital information. We haven't really comprehended the importance of protecting our digital property yet. We'll go to great lengths to protect our physical property, but definitely haven't matured to as strong of a protection with our digital data.

[BSD Mag]: You are a software Developer, a Project Manager and a Security Specialist. Which experience do you find the most useful on the labour market and what do you like to do the most?

[GC]: Overall, I find the security specialist experience the most useful. A strong security background can help you be a great software developer or a great project manager and is useful even in everyday life — because it forces you to look at things from an outsider's point of view and ask - how can I do this better, how can I make this stronger, safer, more secure? Security is an argument that pervades all aspects of an organization and is an important part of our personal safety as well. So, while software development and project management are useful skills, IT security is something that pertains to everyone, everywhere (unless they're living completely off-the-grid, and even then I think I could identify some issues that would affect them).

[BSD Mag]: Where should I start if I wanted to become a security specialist?

[GC]: Before you start, you have to have a passion that borders on obsession. InfoSec more often than not requires a certain mentality - it's not just about having strong technical skills. I don't think there is a real starting point with this industry - it's an epiphany you have to have - start by falling in love with the topic.

[BSD Mag]: What do you think about being experienced in different fields of business and science? Is it more for self-development or it is a must-have skill nowadays? Or maybe it is better to be an expert in a very narrow area?

[GC]: Excellent question and my view on this goes back and forth over the years. Right now, I think that it's better to be an expert in one field. It's better to have certified proof that you know a certain topic well - especially if you're job hunting or have any intention of changing jobs in the future. A certification from a reliable authority makes it that much easier for someone who's hiring to evaluate your skills.

I think that's part of the problem we're seeing today in the job market - it's difficult to accurately measure people's skills. What people write on their resumes and the truth of the matter are often very different realities, even if we're trying to give an honest representation of our skills. Our individual understanding of a degree of knowledge can vary greatly and having a certification, like the CISSP certification that I have, for example, which requires five years of proven experience plus an intense and challenging exam, as well as proof of continuing education credits for as long as you hold your certification, goes closer to an accurate form of measurement. And though I can certainly understand the benefits of having a broad level of knowledge on a variety of topics, I think in the future, when lower-level positions will be automated, very specialized knowledge and reasoning skills - like only humans can do - will be more sought after in the global marketplace.

[BSD Mag]: As you mentioned previously though, having a few different reference points - like having project management experience - can enrich your work. In that case, which field would you recommend for IT security folks, who don't want to diverge too much from their narrow specialty, but would like to get a different perspective nonetheless?

[GC]: Let me specify that, for managerial roles, it's essential to have a broad range of knowledge, you NEED to have an understanding of a wide range of subjects, and even if that knowledge doesn't go deep, it gives you an overall perspective that someone with a single, narrow concentration probably doesn't have.

For those that want to gain perspective, I think the best way would be work on interpersonal relationships and staff education. Most security problems are a result of people not being informed or aware that their behavior is a threat to security. Having the skills to effectively generate awareness - both within the company and with clients - can be a huge security advantage.

[BSD Mag]: What about open source software? Are you an enthusiast of any of them? Do you work with them on a daily basis?

[GC]: Mmmm, open source. I have mixed feelings about open source. I do work with them on a daily basis - Wordpress in particular, I have a lot of experience with. Open source has its pros and cons…I sway towards a negative opinion of open source because so many of them - in order to protect themselves - are written in an extremely convoluted manner. Yes, the code is open, but the developers often incorporate a seeming endless (and, oh-so-vicious) chain of functions that call other functions that call other functions, that if you actually put your hands on the code can be an extremely frustrating and time-consuming experience. If all open source software were written clearly, logically, without unnecessary convolutions and complications, I would be a great proponent of open source. But more often, when I'm dealing with open source, I feel frustrated, disappointed and angry.

[BSD Mag]: If you had a chance, what would you change in open source software to make it less frustrating?

[GC]: I don't think there is a real solution to this problem without changing the very nature of open source, and that would be counter -productive. One of the biggest advantages of open source is that it's free - and if you want that advantage, you have to put up with everything that goes with it. Unfortunately, a lot of the small/medium sized companies that might be turned on by the economical benefits of open source may not realize the limits they're opposing on themselves by doing so: that you accept the software as-is, and unless you're using the software on a very simple level, it's going to cost you more than you might imagine to implement it for your purposes.

An approach to improving open source might be to work towards code standardization - so that programmers can more quickly and easily manipulate the code for their needs or creating better documentation.

[BSD Mag]: I found out that you have experience in Windows NT security implementation. What do you think about all of the problems with Windows 10? Do you work on Windows 10 yourself?

[GC]: Windows, another topic which I feel strongly about - hopefully this interview doesn't make me sound like a grumpy, negative techie. :) About six years ago, I made the switch from Windows to Apple and, though it may sound clichè, I have to say I will never go back. This doesn't mean I no longer interact with Windows, however (unfortunately - haha). I choose not to work on a computer with Windows 10, but I have it installed on Virtual machines for testing and laboratory purposes, etc. And obviously many of our clients use Windows. To give you an idea on how far behind Italy still is - some of the clients I've recently done consulting for were still using Windows Server 2003! Woe is me. I think Windows made some interesting (read: ballsy, egotistical) choices while developing their security policies for Windows 10. It certainly doesn't seem like they're really putting the safety of their clients at the forefront.

[BSD Mag]: What makes Apple better than Windows in that regard?

[GC]: The first thing that makes Apple better than Windows in regards to their putting client safety first, in my opinion, is the App Store. All apps available for download are guaranteed by Apple and reviewed before they are accepted by the store. Apple has extremely high expectations and a strict and complex selection process which filters out malicious developers. If there's ever a problem with an app after it's been added to the store, Apple can quickly remove it. And even if you don't install an app directly from the store, Mac's Gatekeeper protects you from inadvertently installing malicious software.

The App Sandbox in OS X isolates apps from the critical system components of your Mac, your data and other apps and helps ensure that apps only do what they're intended to do.

Would the people in the pictures you publish really be okay with having their image splashed all over the web? The decisions you make about protecting your PII - take a few minutes to really understand how they're protecting your information and your rights, the way you interact with agencies and organizations that have access your personal information. Evaluate your web presence and take steps to make yourself, your family and your friends safer. If you are in IT - share your knowledge about IT security with others in a way that's easy for them to understand.

About Giuseppe:

Giuseppe Canale is an IT professional and entrepreneur based in Northern Italy. Having dedicated over 28 years to the industry, he has the unique advantage of having experienced first hand the evolution of digital technology - starting with the Commodore 64. A Certified Information Systems Security Professional (CISSP) specializing in cyber security, Giuseppe began as a programmer and eventually opened his own IT consulting company in 1997. In recent years he has dedicated himself more specifically to IT security. An avid long-distance runner, when he's not researching the latest threats and vulnerabilities or developing security policies, you'll find him on the track.

The International Consortium of Investigative Journalists, German newspaper Süddeutsche Zeitung, as well as more than 100 other news organizations, have caused an international storm by releasing the Panama Papers this week. There is a very large fly in this ointment however. Why is this 2.6 Tb dataset of over 11 million documents not being released in its entirety?

by Rob Somerville

Just about every mindful adult who has read the news this week concerning the biggest leak in history (so far) will not be at all surprised by the initial content floated in the international media this week. Banks, lawyers, and the men in grey suits behind the scenes facilitated tax avoidance schemes for a privileged few; while the rest of us dutifully pay our taxes – often under the threat of bankruptcy or prison if we default. As the old saying goes, the difference between tax avoidance and evasion is 10 years in jail and a $100,000 fine; yet it is still to be seen if any of the conspirators (both legally and in reality) will face a trip to a dank, damp prison cell. Like a fish rotting from the head down, the stench and level of corruption the Panama Papers exposes turns the stomach. But what has this to do with a technological magazine?

A lot. First, the sheer scale of the dataset is huge. 11.5 million documents. It is larger than many archive collections found in national museums. This would have been difficult to leak 10 years ago, and well-nigh impossible 50 years ago. In 2016, you can fit this all on an external drive smaller than a paperback book, or if you have $1,500 to spare, it almost fits on a couple of 1TB USB drives. The exact technique used to extract and transfer the data is currently unknown (it may have just been FTP'd from the offices of Mossack Fonseca) but any security conscious whistle-blower would be cognizant of the risks no matter how it was transferred. Without the leverage of technology, this incident would not have happened. Was it a hacker? A disgruntled employee? A lawyer with a conscience? Time may tell, but whatever the scenario the perpetrator of this act must have nerves of steel.

Second, there is the ethical and moral considerations to be factored into account. Far be it for me to criticize the ICIJ, who have performed sterling work in bringing this whole shady affair into the spotlight, there is a much bigger question here. While the right to privacy is enshrined in law in most jurisdictions, how can this be reconciled with the damage done to society when the evidence of such large scale hypocrisy and potential criminality – to use an American legal term – is Fruit of the Poisonous Tree?

Two wrongs do not make a right, and it could be argued that as this information was obtained illegally, it should be destroyed. After all, as individuals we would not be very happy if our medical or financial records were released to the press.

What is challenging about the whole scenario is the knife edge walked between the public good and personal interest. No doubt, amongst the data set there will be perfectly innocent information that would cause distress if it were be to be published. This, along with the libel laws, is the main plank of the journalists' case to maintain control over the data while they process it. If given to lawyers and courts to deal with, the time-scales and costs to analyze, review, and prosecute would be so excessive that little would be achieved. The same could be said of national tax agencies. So there is an ironic expediency to have a trial by media, as the quickest way to slide down the greasy pole of power to a severely damaged reputation. However, this introduces a another factor – political and financial bias. No matter how professional the paper, they will have their own agenda, a particular spin they want to put on the ball. And advertisers. It seems terribly coincidental that at a time when the West is going all out to discredit the Russian President, that he has been the main focus of this leak in the Western media, despite his loose connections with the Russian individual named.

If I were a judge, my response to all of this would be to release all of the data into the public domain on the basis that a million pairs of eyes will get to the bottom of this sordid story a lot quicker than any team of journalists, no matter how talented or dedicated. Applications could be written (like Seti at home for instance) to data-mine, find patterns, and extract the relevant facts.

While this might seem like an extreme position; However, if the little detail that has emerged so far is anything to go by, I believe there are many more dirty little secrets buried in these documents. Redressing the amount of damage inflicted on society (especially in these times of austerity) far outweighs any individual right. You might argue that you have done nothing wrong, and I would be the first to argue for your right of privacy and confidentially in normal circumstances. But these are not normal circumstances.

50 years ago, it was the exclusive domain of the safe cracker, spy, thief, and charlatan to gain illicit access to confidential information. While sensitive information was stolen, the physical amount was limited to photocopies or what could be captured on a high-resolution miniature camera. Today, any script kiddie has potential access from his bedroom or basement. Organized crime is leading the charge in capturing data on an unprecedented scale, albeit for their own nefarious ends rather than the public good. Prior to this scoop, we had not seen anything yet on this scale, but give it a few more years and it will be quickly surpassed. More and more data is being migrated to the cloud, and huge datasets can be cloned in minutes. At the press of a key or a click of a mouse, this can be made available, replicated and distributed with ease. As a society, we need to work out a solution to the ethical issues and moral quandaries the actions of whistle-blowers and indeed hackers raise – and quickly. If not, when the next big exposure happens, the journalists and lawyers will still be digging their way through the fine print of the Panama Papers.